A dynamic attack graphs based approach for impact assessment of vulnerabilities in complex computer systems

Antoine Boudermine

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAT046

Thèse de doctorat

TELECOM Paris

IP PARIS

A Dynamic Attack Graphs based approach for Impact Assessment of Vulnerabilities in Complex Computer Systems

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 19 Décembre 2022, par

ANTOINE BOUDERMINE

Composition du Jury :

Pascal Lorenz Professeur, Université de Haute-Alsace	Président
Solange Ghernaouti Professeure, Université de Lausanne	Rapporteuse
Lyes Khoukhi Professeur, Normandie Université	Examinateur
Sherali Zeadally Professeur, Université du Kentucky	Examinateur
Jean Leneutre Maître de conférences, Télécom Paris	Examinateur
Youssef Laarouchi Spécialiste en cybersécurité, EDF R&D	Examinateur
Rida Khatoun Professeur, Télécom Paris	Directeur de thèse
Jean-Henri Choyer Adjoint CERT, NAVAL GROUP	Co-directeur de thèse
Julien Francq Responsable Recherche & Innovation en Cybersécurité, NAVAL GROUP	Invité

Résumé

Les systèmes informatiques sont devenus omniprésents dans nos sociétés modernes. Par exemple, les systèmes d'information permettent de supporter le processus métier de nombreuses entreprises en permettant de collecter, stocker, traiter et distribuer de l'information. Les systèmes industriels permettent quant à eux d'augmenter les performances de production, tout en améliorant les conditions de travail et la qualité des produits fabriqués.

La complexité de ces systèmes n'a cessé de croître des dernières années, avec un nombre toujours plus important de composants aux propriétés diverses et en forte interaction. Les attaques ciblant ces systèmes se sont elles aussi complexifiées et peuvent se dérouler en plusieurs étapes sur une longue période de temps. Par exemple, le comité national démocrate, qui est un organisme responsable du parti démocrate aux États-Unis d'Amérique (USA), a été compromis en juillet 2015 par le groupe d'attaquant APT29 [1]. Les attaquants ont maintenu un accès au système informatique jusqu'au 10 juin 2016, soit presque 1 an après la compromission initiale. De nombreuses données ont été dérobées, incluant des e-mails et des documents confidentiels en rapport avec les élections américaines.

Les organisations ont donc besoin d'évaluer la sécurité de leurs systèmes informatiques afin d'identifier les contremesures qui doivent être mises en place. Cependant, les méthodes de gestion du risque actuellement utilisées, comme EBIOS RM en France ou le NIST RMF aux USA, ne sont plus adaptées à la forte complexité des systèmes et des attaques qui ont lieu de nos jours. En effet, ces méthodes reposent sur un travail humain trop important, ce qui rend l'analyse de risque peu précise et sujette aux erreurs. L'objectif est donc de proposer une méthode d'analyse de risque plus automatisée et capable de gérer des systèmes complexes de plusieurs milliers de composants.

Pour atteindre cet objectif, il faut être en mesure de modéliser le plus précisément possible le système ainsi que les capacités de l'attaquant, malgré les informations parfois limitées ou

incertaines dont on dispose en donnée d'entrée. Il est par exemple difficile d'identifier l'ensemble des faiblesses présentes dans un système ou d'évaluer la difficulté avec laquelle un attaquant parviendra à les exploiter. La modélisation du système est d'autant plus compliquée que cette dernière doit être la plus exhaustive possible afin de garantir que l'attaquant ne parviendra pas à identifier un moyen de compromettre le système que les défenseurs n'avaient pas envisagé. La méthode d'analyse de risque doit également pouvoir être appliquée dans des systèmes de très grande taille en un temps raisonnable, afin que les résultats obtenus puissent être utilisés par les équipes de sécurité en s'inscrivant dans un processus d'amélioration continue de la sécurité du système.

Des travaux de recherche ont déjà été réalisés pour tenter de répondre à cette problématique. La méthode la plus couramment utilisée consiste à modéliser l'ensemble des chemins d'attaque qui permettent à un acteur malveillant d'atteindre ses objectifs de compromission du système. Un chemin d'attaque représente donc une succession d'actions réalisées par l'attaquant et de conséquences de ces dernières sur le système. Ces chemins sont généralement représentés sous la forme d'un graphe appelé graphe d'attaque. Ce graphe permet de modéliser les différentes actions qui permettent d'atteindre un objectif intermédiaire de l'attaquant, ainsi que les différentes conditions nécessaires à la réalisation d'une action. Cette représentation permet de modéliser d'une manière compacte l'ensemble des chemins d'attaque.

Les graphes d'attaque peuvent ensuite être utilisés pour évaluer les chances de succès de l'attaquant en tenant compte de la difficulté des actions réalisées. Pour ce faire, il est possible d'évaluer les chances de succès de chaque chemin d'attaque. Cependant leur nombre évolue de façon exponentielle par rapport à la taille du graphe d'attaque, ce qui rend cette solution inapplicable dans des systèmes de grande taille. C'est pour cette raison qu'une solution plus efficace consiste à calculer directement la probabilité d'être VRAI pour chaque nœud du graphe d'attaque. Pour les nœuds représentant une action de l'attaquant, il s'agit de la probabilité de la réaliser, tandis que pour un nœud représentant une conséquence d'une attaque, il s'agit de la probabilité que l'attaquant parvienne à atteindre cet objectif.

Le calcul de ces probabilités peut se faire directement à partir du graphe d'attaque ou en convertissant ce dernier en un modèle plus générique comme un réseau Bayésien ou un réseau de Petri. L'avantage de ces modèles est que de nombreux algorithmes existent déjà et peuvent être utilisés pour calculer les chances de succès de l'attaquant. Cependant, aucune de ces méthodes de calculs n'est adaptée aux systèmes complexes, qu'il s'agisse des calculs réalisés directement dans un graphe d'attaque, des calculs d'inférences dans un réseau Bayésien ou de la génération du graphe d'accessibilité dans les réseaux de Petri. En effet, la complexité en temps dans le pire des cas de ces algorithmes ne permet pas d'analyser la sécurité de systèmes composés

de plusieurs milliers de composants. Il est cependant possible d'utiliser des algorithmes pour approximer les chances de succès de l'attaquant en un temps raisonnable. Par exemple, il est possible d'échantillonner les variables aléatoires présentent dans un réseau Bayésien ou de réaliser des simulations dans un réseau de Petri.

D'autres solutions ont été proposées pour évaluer l'impact d'une attaque sur le système. Il est possible d'utiliser un modèle épidémiologique pour étudier la propagation d'un virus informatique ou l'impact d'une attaque à travers un système. L'inconvénient de cette solution est qu'il n'est pas possible d'étudier la propagation de plusieurs impacts de différentes natures au même instant. Il n'est par exemple pas possible de modéliser une attaque combinant la prise de contrôle distant d'un composant et l'indisponibilité d'un service réseau.

La théorie des réseaux complexes peut être utilisée pour mesurer l'impact d'une attaque à partir d'une étude topologique du réseau. Les nœuds représentent des composants du système tandis que les liens représentent des connexions. L'indisponibilité d'un composant suite à une attaque est représentée en supprimant le nœud du réseau. Des métriques sont ensuite définies pour mesurer les performances du système en fonction des nœuds supprimés. Cette méthode est utilisée pour identifier les composants critiques d'un système afin de prioriser les mesures de protection sur ces derniers. Cependant, il est uniquement possible de modéliser l'indisponibilité d'un composant comme conséquence d'une attaque.

Les automates permettent de modéliser l'évolution de l'état d'un système en fonction des actions réalisées par l'attaquant et des changements d'états internes du système. Des algorithmes de *model checking* sont ensuite utilisés pour mesurer les conséquences de l'attaque. Cependant, ce modèle souffre d'une explosion combinatoire du nombre d'états possibles du système et n'est donc pas applicable lorsque ce dernier est composé de milliers de composants.

La plupart des solutions proposées ne tiennent pas compte du comportement dynamique du système lors de l'évaluation du risque d'une cyber attaque. Pourtant l'évolution du système au cours du temps peut avoir un impact sur les chances de compromission par un acteur malveillant. Par exemple, la pratique de plus en plus courante du télétravail a pour effet que les postes informatiques des collaborateurs sont régulièrement déplacés entre le réseau interne de l'entreprise et le réseau domestique de l'utilisateur. Un acteur malveillant pourrait profiter que l'ordinateur soit connecté au réseau domestique qui est peu sécurisé pour compromettre la machine, puis attendre que cette dernière soit de nouveau connectée au réseau interne de l'entreprise pour pouvoir se propager à travers le système informatique.

Nous avons donc eu pour objectif de proposer une solution d'analyse de risque qui tient compte du comportement dynamique du système et qui peut être appliquée lorsque ce dernier

est composé de plusieurs milliers de composants.

Notre première contribution a été de proposer une solution permettant de construire un graphe d'attaque à partir d'une représentation incluant les propriétés dynamiques du système à analyser. Nous avons ensuite montré que notre modèle permettait d'identifier davantage de chemins d'attaque qu'un graphe d'attaque basé sur une représentation statique du système.

Notre seconde contribution a été de proposer un algorithme permettant de simuler des attaques à partir du graphe d'attaque dynamique précédemment construit. Ces simulations permettent d'approximer les chances de réussite de l'attaquant en tenant compte de la difficulté des actions réalisées et de l'incertitude relative à certaines propriétés du système. Une première métrique a été définie pour mesurer l'évolution de la probabilité qu'un nœud du graphe d'attaque soit VRAI au cours du temps. Pour un nœud représentant une action, il s'agit de la probabilité que l'attaquant parvienne à la réaliser à un instant t. Pour un nœud représentant la conséquence d'une action, il s'agit de la probabilité que l'attaquant parvienne à atteindre cet objectif à partir d'un instant t. Une seconde métrique permet de mesurer le temps nécessaire pour que la probabilité qu'un nœud du graphe d'attaque soit VRAI dépasse une valeur seuil p_s définie par l'utilisateur. Cette métrique peut être utilisée pour afficher une *heatmap* du système en montrant les composants les plus impactés par l'attaque. Enfin, une troisième métrique a été définie pour mesurer la différence entre deux simulations d'attaque. Cette métrique peut être utilisée pour mesurer l'impact d'un changement de configuration du système, comme la correction d'une vulnérabilité, sur les chances de succès de l'attaquant.

Nous avons ensuite montré que notre solution pouvait être utilisée dans des systèmes complexes. La complexité en temps dans le pire des cas de tous les algorithmes utilisés a été évaluée. Plusieurs benchmarks ont également été réalisés pour évaluer les performances réelles de nos algorithmes. Les résultats montrent que le temps de génération du graphe d'attaque évolue de façon quadratique par rapport au nombre de machines dans le système, tandis que le temps de réalisation d'une simulation d'attaque évolue linéairement par rapport à la taille du graphe d'attaque. Nous avons ensuite appliqué notre méthode sur un système informatique complexe composé de 60,000 postes utilisateurs. Nous avons utilisé une machine virtuelle Ubuntu 20.04 avec un noyau Linux 5.15.0-41-generic, 16 CPUs à 2GHz et 64 GB de mémoire. Si toutes les simulations d'attaque sont réalisées en parallèle, il faut approximativement 12 heures pour générer le graphe d'attaque dynamique et pour calculer les probabilités de réussite de l'attaquant.

Pour conclure, notre solution permet d'analyser la sécurité dans des systèmes complexes tout en tenant compte de leurs propriétés dynamiques. Les métriques qui ont été définies permettent de mesurer l'évolution de la probabilité de compromission des composants du système,

d'afficher une *heatmap* des composants les plus impactés par l'attaque ainsi que de mesurer les conséquences d'un changement de configuration du système sur les chances de succès de l'attaquant. Cependant, nous restons limités à l'analyse de systèmes composés de moins de 100,000 éléments à cause de l'évolution quadratique du temps de génération du graphe d'attaque dynamique. Il serait intéressant de vérifier s'il n'est pas possible de générer ce graphe d'une manière plus efficace. De plus, la modélisation du système qui est nécessaire à la construction du graphe d'attaque, n'est pas automatisée et reste donc une étape difficile à réaliser par des humains, surtout dans des systèmes complexes. Des solutions d'automatisation devraient être recherchées et une Interface Homme-Machine (IHM) devrait être développée pour faciliter le travail collaboratif entre les acteurs responsables de la sécurité du système informatique. Notre solution pourrait être utilisée à l'avenir pour identifier les mesures protectrices à mettre en place en priorité par les équipes de sécurité. En prenant le point de vue opposé de l'attaquant, cette solution pourrait permettre d'identifier les chemins d'attaque optimaux que les équipes offensives devraient utiliser pour atteindre leurs objectifs.

Remerciements

Je remercie tout d'abord M. Rida Khatoun, mon directeur de recherche, pour l'encadrement qu'il a mis en place durant cette thèse de doctorat et qui m'a permis de travailler avec méthodologie tout en étant en forte autonomie.

Je tiens à remercier toute l'équipe encadrante de Naval Group avec qui, sans leur soutien constant, cette thèse de doctorat n'aurait jamais vue le jour. Je remercie tout particulièrement M. Pascal Mercier, directeur du CERT Naval Group et M. Jean-Henri Choyer, co-directeur du CERT Naval Group, pour m'avoir accordé leur confiance le jour où ils m'ont confié le soin de réaliser cette thèse de doctorat, ainsi que pour leur soutien sans faille et leurs encouragements qui m'ont permis de surmonter les nombreux obstacles et moments difficiles menant au titre de docteur.

Je tiens également à remercier M. Julien Francq, responsable recherche et innovation en cybersécurité à Naval Group, pour son accompagnement et ses précieux conseils qui m'ont permis de grandement améliorer la qualité de mes travaux de recherche.

Mes remerciements vont également à toute l'équipe du CERT Naval Group pour leur soutient et leur bonne humeur au quotidien, qui permet de créer une ambiance de travail exceptionnelle dans laquelle on s'épanouie personnellement et professionnellement.

Pour finir, mes remerciements vont à toute ma famille et plus particulièrement à mes parents avec qui, sans leur soutient et leur persévérance depuis déjà plus de 27 ans, je ne serais pas devenu la personne que je suis aujourd'hui.

Table of contents

List of Figures

List of Tables

Introduction

Context and motivation

Computer systems have become ubiquitous in our societies. They allow us to exchange information across the world, to connect everyday objects to make them more intelligent or to improve the productivity of our industries. In [2], the authors defines a system as a set of interacting units or elements that form a whole in order to perform some functions. The purpose of computer systems is to interconnect electrical components in order to perform certain tasks. The precise properties of this type of system will be given in the next section. Over time, the complexity of these systems has continued to grow. We understand by complex any system which corresponds to the definition given in [3], meaning a system made up of a large number of elements in strong interaction and with diverse characteristics, and whose evolution is sensitive to small perturbations. It is therefore very difficult to predict the impact that a vulnerability could have on the evolution of the computer system's state when exploited by a malicious actor. We will see in the next section how modern computer systems can be considered as complex systems.

Computer systems are also increasingly used in critical environments. They can be found, for example, in the management systems of nuclear power plants, in the autopilot system of some vehicles or in the weapon systems of modern warships. It is obvious that the malfunction of one of these systems can lead to serious consequences, such as death or injury, destruction of important equipments or negative impact on the environment.

However, vulnerabilities are regularly discovered on some components of computer systems. These may be weaknesses related to a configuration flaw, improper architectural design, or a problem with the implementation or use of the solution. Databases, such as the one provided by National Vulnerability Database (NVD), publicly list a large number of vulnerabilities that have been discovered in widely deployed components of computer systems.

The exploitation of these vulnerabilities by a malicious actor can therefore have serious consequences. During an attack, an attacker can perform a succession of actions that allow him to reach his final goal. To achieve this goal, the attacker uses a variety of tactics. For each of these tactics, there are a multitude of attack techniques to achieve an intermediate goal of the malicious actor. The matrix MITRE ATT&CK [4], visible in Figure 1, allows to make the link between the tactics used by the attacker and the techniques allowing to realize them.

Figure 1: MITRE ATT&CK matrix making the link between the tactics used by the attackers and the techniques employed to achieve them

The impacts that an attack can have on system components are generally summarized as loss of availability, integrity or confidentiality. The need for availability can be defined as the property of accessibility at the desired moment of the assets by authorized persons (for example, the asset must be available during the expected usage periods). The need for integrity can be defined as the property of accuracy and completeness of assets and information (for example, an illegitimate modification of an asset must be detected and corrected). Finally, the need for confidentiality can be defined as the property of assets to be accessible only to authorized people.

Malicious actors are often categorized into three groups [5]:

- **the commodity threats.** Most of the time, these are cybercriminal groups whose objective is to make money. The two most commonly used *modus operandi* are: (1) exfiltration of sensitive data and request for a ransom to keep them from being made public. Attackers can in some cases make part of the stolen information public to prove their crime and put the organization under pressure. (2) The encryption of company data and the request for a ransom to recover them;
- **hacktivism.** These are people with a political agenda who target specific organizations to try to destabilize them or affect their reputation. The most commonly used *modus operandi* is to exfiltrate sensitive information and make them public;
- **the Advanced Persistent Threat (APT).** These are groups of attackers with significant human and financial resources that allow them to achieve specific objectives. The attacks target a specific organization and the objective is to maintain access to the system over a long period of time and remain undetected.

The different tactics of these attacking groups are summarized in Figure 2.

Many cyber attacks have already taken place in the past with sometimes disastrous consequences. These attacks have often targeted information technology systems. In May 2017, the WannaCry ransomware spread across the globe, affecting nearly $300,000$ systems including hospitals and governments [6]. This worm used a 0-day named Eternal Blue to propagate itself. A 0-day is a vulnerability that has not yet been published or has no known patch. More recently in 2019, malware was injected into the SolarWinds solution and spread through customer updates [7]. This malware allows to create a backdoor in the system and gives remote access to the attackers. More than $18,000$ customers were affected, including critical U.S. organizations such as the Treasury Department, a nuclear weapons design and production laboratory, and the government.

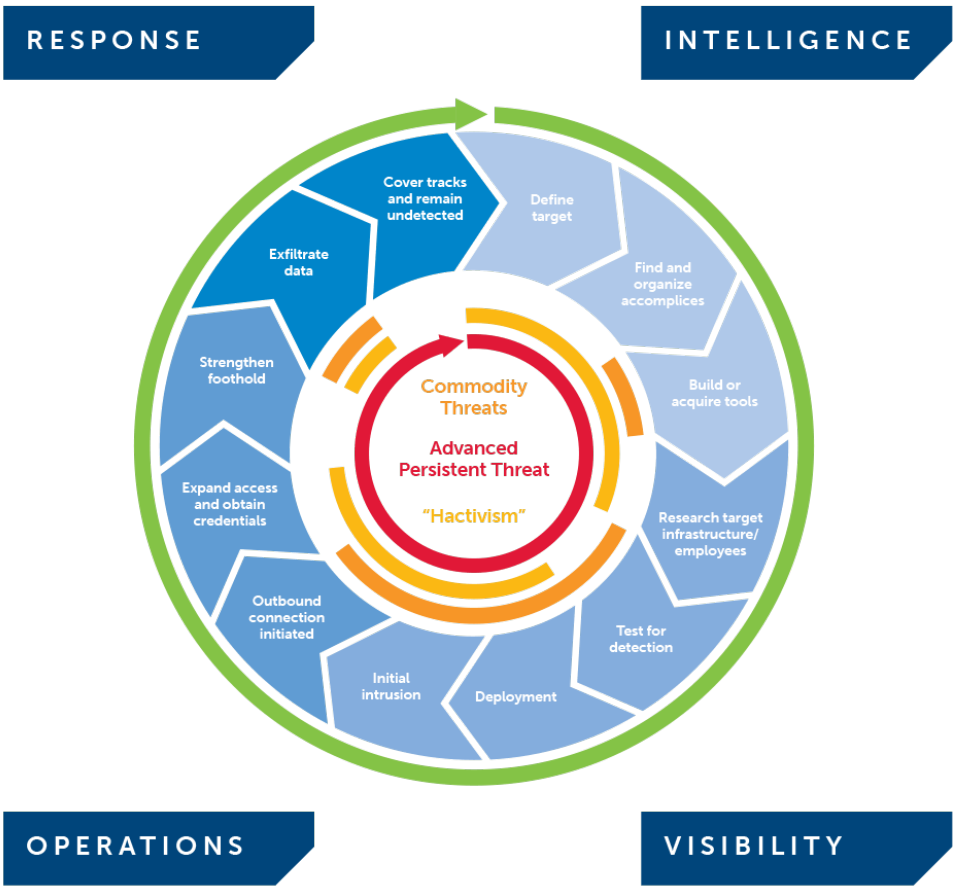


Figure 2: Tactics, Techniques and Procedures (TTP) used by different types of attacker groups

Source: "Advanced Persistent Threats: Learn the ABCs of APTs - Part A." (), [Online]. Available: https://www.secureworks.com/blog/advanced-persistent-threats-apt-a

Cyber-physical systems have not been spared. On December 23, 2015, several control centers of the Ukrainian power grid were targeted by an attack resulting in a power outage for 6 hours in the capital and its surroundings [9]. Hundreds of thousands of consumers were impacted in the heart of winter. This attack took place in a context of civil war and high tension with Russia, from where the attack was probably launched. The computer worm named Stuxnet and discovered in 2010 was used to destroy several centrifuges of a uranium enrichment plant in Iran [10]. The worm was spread via Universal Serial Bus (USB) device and targeted Windows-based computer workstations used for automation and supervision of electromechanical equipment.

It is possible for organizations to assess the impact of these vulnerabilities at the component

level, but there is a real difficulty in doing the same at the level of an entire system. Nevertheless, this assessment is essential for an organization to be able to prioritize its remediation actions, or to defer them when corrective actions cannot be taken immediately without seriously affecting the system.

First, the organization must be able to identify the vulnerabilities present on its computer system. To do this, it is possible to use a vulnerability scanner such as Nessus [11] which will search for vulnerabilities on the various network services or locally on the machines. However, these scanning tools can only find vulnerabilities that have been publicly reported such as those present in databases such as the Common Vulnerabilities and Exposures (CVE) maintained by the MITRE organization. But as we saw with the WannaCry attack, some vulnerabilities are not shared when discovered and are used by attacker groups. Furthermore, these tools will not be able to find vulnerabilities on solutions developed internally in the organization, nor vulnerabilities related to a misconfiguration such as the presence of a weak password associated with an administration account. Organizations should therefore also perform security audits conducted by experts to detect this type of vulnerability. This may include, for example, offensive tests performed by pentesters or application code reviews. However, these operations have a significant and recurring financial cost for the company.

Remediation of identified vulnerabilities can also be a technical and operational challenge. Sometimes security patches are not available, either because it is a 0-day vulnerability or because the vulnerability concerns an internally developed solution. It is still possible in some cases to apply work-arounds to limit the impact of a vulnerability for which no patch is available. When a vulnerability concerns an internally developed product, the organization must develop a patch itself, which can be costly in time and money.

Even when a patch is available, it is not always possible to apply it. For example, if the vulnerability concerns a critical application of the organization and the implementation of the patch requires an interruption of the service, it could have serious consequences to apply it and the system managers may prefer to delay the correction. Even more constraining, a warship in operation will not interrupt its mission if the vulnerability is not considered critical. In this case, the decision to fix the vulnerability depends largely on the impact of the vulnerability on the proper functioning of the system.

Computer system managers therefore need a method to assess the security of their system and to measure the criticality of a vulnerability in relation to its impact on the organization. Several risk management methods already exist and are used in practice. The EBIOS RM method [12] proposed by the French National Agency for Information Systems Security (ANSSI)

is used to manage the risk of an attack based on:

1. sources of risks and their objectives;
2. strategic scenarios representing the attack paths that allow a risk source to achieve its objective. These attack paths are realized at the business level of the studied objects and are evaluated in terms of severity;
3. operational scenarios representing the techniques used by the risk sources to achieve the strategic scenarios. These scenarios focus on the third party supports of the system and are evaluated in terms of likelihood;
4. synthesis of the previously identified risks and definition of a risk management strategy. In addition, definition of what the residual risks will be.

In the United States of America (USA), the Risk Management Framework (RMF) [13] proposed by the National Institute of Standards and Technology (NIST) is divided into 7 different steps:

1. **prepare:** list all the assets we want to protect and perform a organisation-level risk assessment. NIST Special Publication 800-30 [14] presents the methodology for assessing risk in an information system. This process, visible in Figure 3, consists of 6 different steps:
   (a) identification of the threat sources that the organization must face;
   (b) identification of the threat events that these threat sources can perform;
   (c) identification of vulnerabilities in the organization that can be exploited by threat sources to carry out their threat events, as well as the preconditions that must be met;
   (d) assess the likelihood that a threat source will initiate and succeed in carrying out a threat event;
   (e) assess the impact of a successful threat event on the organization's assets and processes;
   (f) assess information security risks based on the likelihood of vulnerability exploitation and the impact on the organization, and by considering the uncertainty of the risk assessment.
2. **categorize:** classify the system and information according to the impact analysis.;
3. **select:** define protection measures to be put in place to protect these assets;

4. **implement:** implement the previously defined protective measures;
5. **assess:** test that the protective measures implemented are adequate to protect the assets;
6. **authorize:** accept that some risks remain;
7. **monitor:** continuously monitor the relevance of the measures implemented in the system.

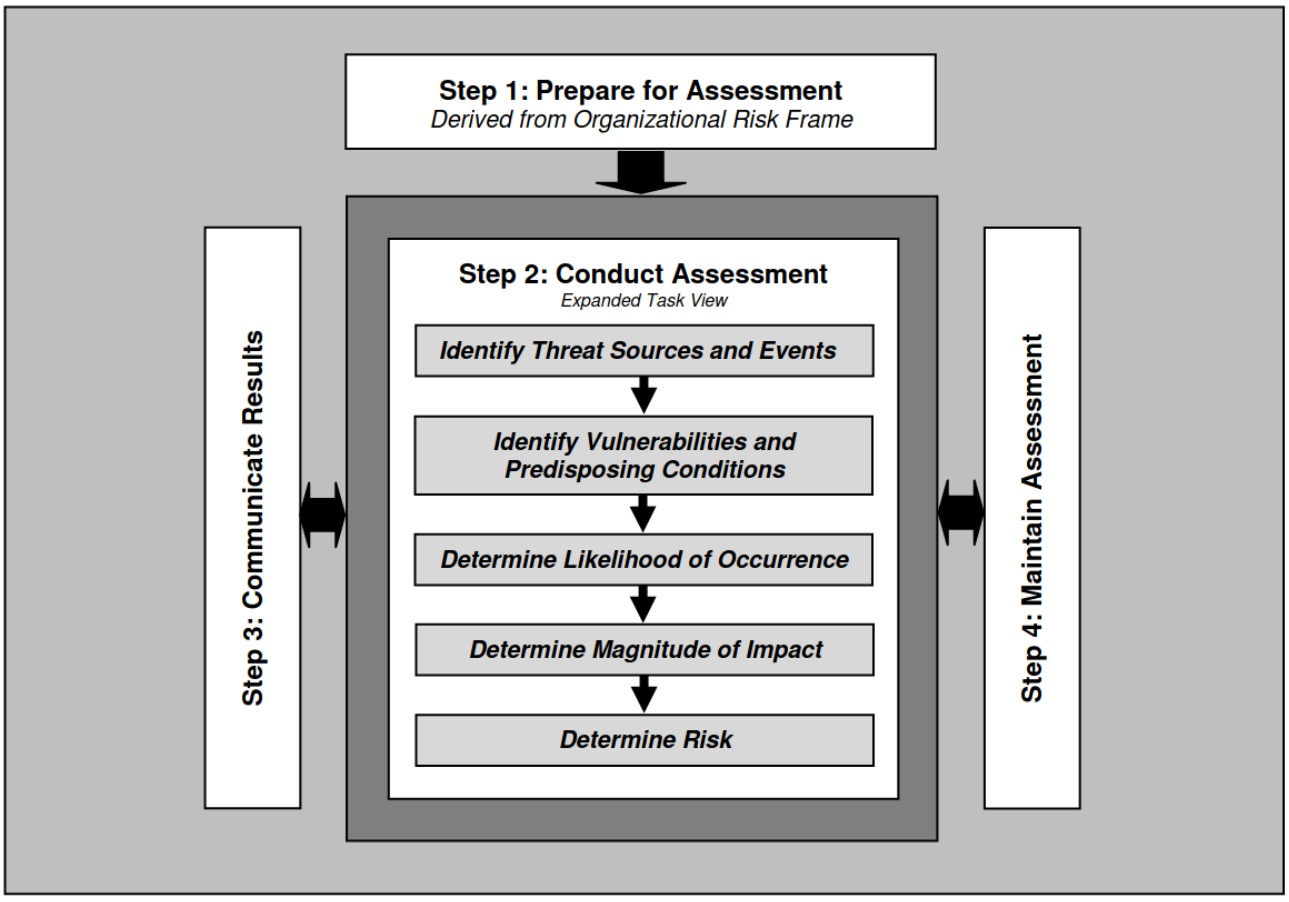


Figure 3: NIST risk assessment process

Source: "Guide for Conducting Risk Assessments." (), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf

The problem with the above-mentioned methods is that they mainly rely on the work of experts. This human work has several disadvantages. First, the risk assessment is subjective. Experts must qualitatively assess the likelihood and impact of risk scenarios, without being able to rely on quantitative and objective calculations. The EBIOS method defines so-called strategic and operational scenarios, which often remain at a high level without giving details of the different actions that can be carried out by the attacker. The definition of these attack paths by experts is often based on fictitious scenarios and on few real data coming from the

system. For example, a step in an attack scenario might be the injection of malicious code into a website, but without specifying the technical details or the vulnerabilities exploited to achieve this. On the other hand, risk assessment methodology proposed in the NIST RMF process does not provide specific guidance on how vulnerabilities are identified, nor how to assess their likelihood and impact. Finally, both risk assessment methods lack scalability. The large number of attack paths to define in the EBIOS method cannot be performed by humans, while the number of vulnerabilities to analyze in the RMF method becomes too large.

In order to solve the problem of assessing the impact of a vulnerability, the Common Vulnerability Scoring System (CVSS) has been defined based on several criteria that measure the difficulty and impact of the exploitation. The different criteria used to measure the difficulty of the exploitation are:

- **AttackVector:** the context in which the vulnerability is exploitable (Network, Adjacent Network, Local, Physical);
- **AttackComplexity:** requirements to perform the exploitation of the vulnerability (Low, High);
- **PrivilegeRequired:** level of privileges required to exploit the vulnerability (None, Low, High);
- **UserInteraction:** action required from the user to exploit the vulnerability (None, Required).

The criteria for measuring the impact of a vulnerability are:

- **confidentiality:** impact of the vulnerability on the confidentiality of the asset (None, Low, High);
- **integrity:** impact of the vulnerability on the integrity of the asset (None, Low, High);
- **availability:** impact of the vulnerability on the availability of the asset (None, Low, High).

The Base Score (BS) in version 3.1 is calculated with equation (1). The calculation of this score requires the calculation of two sub scores. The Impact Sub Score (ISC) is calculated with equation (2), while the Exploitability Sub Score (ESC) is calculated with equation (3). The change of scope means that the exploitation of a software's vulnerability impacts resources that are beyond its privileges.

$$\begin{aligned}
&\text{If } (ISC \leq 0):\ BS = 0 \\
&\text{If (Scope Unchanged):}\ BS = Roundup(Min[(ISC + ESC), 10]) \\
&\text{If(Scope Changed):}\ BS = Roundup(Min[1.08 \times (ISC + ESC), 10])
\end{aligned} \tag{1}$$

$$\begin{aligned}
&\text{If(Scope Unchanged):}\ ISC = 6.42 \times ISC_{base} \\
&\text{If(Scope Changed):}\ ISC = 7.52 \times [ISC_{base} - 0.029] - 3.25 \times [ISC_{Base} - 0.02]^{15} \\
&\text{with } ISC_{Base} = 1 - [(1 - Impact_{Conf}) \times (1 - Impact_{Integ}) \times (1 - Impact_{Avail})]
\end{aligned} \tag{2}$$

$$ESC = 8.22 \times AttackVector \times AttackComplexity \times PrivilegeRequired \times UserInteraction \tag{3}$$

The CVSS score also allows to take into account criteria that evolve over time, such as the presence of exploit code, a patch from the editor or confidence in the actual existence of the vulnerability. The following criteria are defined:

- **exploit code maturity:** existence of an exploit or attack code in progress (Not Defined, Unproven that exploit exists, Proof of concept code, Functional exploit exists, High);
- **remediation level:** existence of a remediation method such as a workaround or a patch from the editor (Not Defined, Official fix, Temporary fix, Workaround, Unavailable);
- **report confidence:** measures the degree of certainty that the vulnerability exists (Not Defined, Unknown, Reasonable, Confirmed).

Equation (4) shows the calculation of the CVSS 3.1 Temporal Score (TS).

$$TS = Roundup(BaseScore \times ExploitCodeMaturity \times RemediationLevel \times ReportConfidence) \tag{4}$$

The environmental impact can also be considered. The following criteria are defined:

- **security requirements:** the security requirements of the defender in terms of availability, integrity, and confidentiality (Not Defined, High, Medium, Low);

- **modified base score metrics:** vulnerability characteristics in the context of the defender.

The CVSS score has the advantage of being used by many actors and is already associated with each CVE in database. However, this score only measures the impact of a vulnerability at the asset level, but not at the system one.

Scientific challenges

We have shown that the risk management solutions currently used to assess the security of computer systems are no longer adequate and we have highlighted the necessity of developing a solution that meets the following requirements:

1. the method must be applicable in any kind of computer system;
2. the method must be able to be used in complex and dynamic systems. In this thesis, we set ourselves the objective of analyzing the security of systems composed of several thousands of elements;
3. the calculations made to assess risk must be as objective as possible. To do this, the security analysis must be based on real information directly issued from the studied system;
4. the risk assessment process should be automated as much as possible to reduce the amount of work that needs to be done by humans and to avoid errors;
5. the risk analysis method must also be able to assess the impact of one or a set of vulnerabilities on the system.

To meet the above requirements, several scientific challenges must be overcome. First of all, we must succeed in modeling the system as well as the capabilities of the attacker. Although a model is a simplification of reality, it must allow the most accurate representation of the various properties of the components and their interactions. It is also possible to model the evolution of the system over time. The difficulties related to the modeling come from the limited information that can be obtained, either about the system or the attacker. Concerning computer systems, this information can for example come from network filtering rules present on firewalls or from network scanning results. However, some information may remain unknown or uncertain, and the model must be able to take this into account. For example, the time required for a malicious actor to exploit a vulnerability cannot be known with sufficient confidence.

Difficulties related to modeling can also come from the model itself. Indeed, the size of the model as well as the time required for its construction must be adapted to the representation of complex systems, which in this thesis means systems that can be composed of several thousands of elements.

There are also difficulties in assessing the risk of compromise and measuring the impact of vulnerabilities. Indeed, the calculations performed must be adapted to models representing complex systems. It is therefore necessary to be vigilant about the complexity of the algorithms used in order to guarantee that the calculations carried out finish in a reasonable amount of time.

Objectives and main contributions

The objective of this thesis is to propose a risk analysis method applicable to complex computer systems of various nature. This method should make it possible to measure the risk of compromise of system components in a more precise and automated way than the methods currently used, such as EBIOS RM or NIST RMF. Moreover, our method must be able to measure the impact of one or a subset of vulnerabilities on the proper functioning of the system. This assessment must take into account the structure and properties of the organization's system, unlike the CVSS score, which assesses the impact of a vulnerability only at the component level.

Several works have already been done in this research area in order to model the attack paths allowing a malicious actor to compromise the system and to evaluate its chances of success. These attack paths are represented in the form of a graph called an attack graph. However, the modeling of attack paths is based on a static representation of the system. It does not allow to consider dynamic behaviors such as the changes of network connections of the user workstations alternatively working at home and in the office. These permanent changes in the network topology can however have an impact on how a malicious actor will penetrate the system. Our objective is therefore to propose a new method for generating attack graphs that is based on a dynamic representation of the system.

The complexity of the calculations performed to evaluate an attacker's chances of success based on attack paths is rarely adapted to complex systems. Our objective is to propose a method to compute the attacker's success probabilities from the previously constructed dynamic attack graph. The calculations carried out must allow the analysis of systems composed of several thousands of components and to be completed in a reasonable amount of time. In this

thesis, we decided that the risk assessment should not last longer than one week in order to give the security teams time to apply the necessary countermeasures before performing a new analysis.

Finally, our goal is to propose a metric to measure the impact of one or a subset of vulnerabilities based on our risk assessment method.

The main contributions of this Ph.D thesis are as follows:

1. analytical study of the existing solutions;
2. modeling cyber attacks in complex systems by considering their dynamic behaviours;
3. calculate the risk of system compromise by malicious actors;
4. measure the impact of a system configuration change, such as the correction of a vulnerability, on the chances of compromise by the attacker;
5. implement and validate our solutions on a complex use case.

Manuscript outline

In the first chapter, we present the results of the literature review on the problem of risk assessment in complex computer systems. This step allows us to identify the limitations of the currently proposed solutions and to define several research questions to guide our work in this thesis.

In the second chapter, we present the work done to answer the research questions defined in Chapter I. The first part of this chapter presents our method for generating attack graphs based on a dynamic representation of the system. In the second part of this chapter, we propose a method for calculating the risk of compromise of the system based on the dynamic attack graph previously constructed. The third part of this chapter aims at showing that our solution is usable in complex systems. To do so, we first evaluate the worst-case time complexity of all the algorithms used. Then we measure the real performance of our algorithms by performing several benchmarks. Finally, we present the results of the test of our method on a complex system. In the last part of Chapter II, we discuss the results obtained by highlighting the advantages and limitations of our work, and by presenting perspectives for future work.

The third chapter concludes this manuscript by outlining how our work addresses the research questions identified in Chapter I.

Chapter I

Review of the literature

I.1 Introduction

The objective of this section is to perform a literature review based on the problematic formulated in the introduction of this thesis. This review allowed us to identify the work that has been completed so far in the field of risk analysis in complex systems. The solutions were divided into different categories according to the methods used:

- methods based on the modeling of attack paths;
- methods using the complex network theory;
- methods applying epidemiological models on computer systems;
- methods based on the modeling of the system state.

Metric		**Score**		
System Modeling	Impact Assessment - IA (Number of criteria)	$IA \leq 1$	$1 < IA \leq 4$	$IA > 4$
	Cascading Effects - CE (Modeling capacities)	None (N)	Limited (L)	Unlimited (U)
	Multi-step Attacks - MA (Modeling capacities)	None (N)	Limited (L)	Unlimited (U)
	Dynamic Properties - DP (Modeling capacities)	None (N)	Limited (L)	Unlimited (U)
	Stochastic Properties - SP (Modeling capacities)	None (N)	Limited (L)	Unlimited (U)
	Uncertainty - UN (Modeling capacities)	None (N)	Limited (L)	Unlimited (U)
Complexity	Modeling Complexity - MC (Maximum system size)	$MC \leq 100$	$100 < MC \leq 1000$	$MC > 1000$
	Risk Assessment Complexity - RAC (Maximum system size)	$RAC \leq 100$	$100 < RAC \leq 1000$	$RAC > 1000$

Table 1: Criteria used for the evaluation of articles

The relevance of each solution in addressing the research problem was assessed and we were

able to highlight the advantages and disadvantages of each of them. Criteria defined in Table 1 have been used to evaluate the different solutions:

- **Impact Assessment (IA):** the ability of the model to represent the diversity of the impact that an attack can cause to the system. The possible values of this metric are:
  - $IA \leq 1$: the model allows to represent only one type of impact;
  - $1 < IA \leq 4$: the model allows to represent up to four different types of impacts;
  - $IA > 4$: the model allows to represent more than four impacts.
- **Cascading Effects (CE):** the ability of the model to represent the cascading effects that can be triggered during an attack. These cascading effects are due to the interdependencies present between the different components of the system. For example, when a component becomes unavailable due to an attacker's action, all the components that depend on its availability will also be impacted by the attack. The possible values of this metric are:
  - None (N): cascading effects cannot be represented in the model;
  - Limited (L): some cascading effects can be represented, such as system interdependencies related to component availability;
  - Unlimited (U): all cascading effects can be represented in the model, regardless of the nature of the interdependencies between the system components.
- **Multi-step Attacks (MA):** the ability of the model to represent multi-step attacks that can be performed by a malicious actor in the system. The possible values of this metric are:
  - None (N): multi-step attacks cannot be represented in the model;
  - Limited (L): some multi-step attacks can be represented, such as the propagation of a virus that always uses the same vulnerability to propagate;
  - Unlimited (U): all multi-step attacks can be represented in the model, regardless of the techniques used or the vulnerabilities exploited by the attacker.
- **Dynamic Properties (DP):** the ability of the model to represent the dynamic behavior of the system, such as representing changes in network topology or the internal state of components. The possible values of this metric are:
  - None (N): the dynamic behavior of the system cannot be represented in the model;
  - Limited (L): some dynamic behaviors of the system can be represented;

  - Unlimited (U): all the dynamic behaviors of the system can be represented in the model.

- **Stochastic Properties (SP):** the ability of the model to represent the random behavior of the system or the attacker, such as changes in the network connections of users' computers when they move or the chances of success of the actions performed by the attacker. The possible values of this metric are:
  - None (N): the random behavior of the system cannot be represented in the model.
  - Limited (L): some random behavior of the system can be represented;
  - Unlimited (U): all the random behaviors of the system can be represented in the model.
- **Uncertainty (UN):** the ability of the model to represent uncertainty about the actual state of the system or about the actual capabilities of the attacker. This uncertainty can for example be modeled by using intervals of values or probability distributions to describe some properties of the system whose value is not known with sufficient confidence. The possible values of this metric are:
  - None (N): uncertainty about the actual state of the system cannot be represented in the model;
  - Limited (L): some uncertainties about the actual state of the system can be represented, such as using a probability distribution to describe the chances of success of the attacker's actions;
  - Unlimited (U): all uncertainties about the actual state of the system can be represented in the model.
- **Modeling Complexity (MC):** the ability of the model to represent complex systems. The possible values of this metric are:
  - $MC \leq 100$: the model allows to represent systems whose size is lower than 100 components;
  - $100 < MC \leq 1000$: the model allows to represent systems with a size between 100 and 1,000 components;
  - $MC > 1000$: the model allows to represent systems whose size is beyond 1,000 components.
- **Risk Assessment Complexity (RAC):** the ability of the model to allow the assessment of the risk of compromise by a malicious actor in complex systems. The possible values of this metric are:

- $RAC \leq 100$: the model allows to assess the risk of compromise in systems whose size is less than 100 components;
- $100 < RAC \leq 1000$: the model is used to assess the risk of compromise in systems with a size between 100 and 1,000 components;
- $RAC > 1000$: the model allows to assess the risk of compromise in systems whose size is beyond 1,000 components.

The evaluation of the solutions allowed us to identify some limitations in their ability to fully address the research problem of this thesis. Research questions have been formulated from the identification of these limitations in order to define future work to be done in the continuation of this thesis. In the rest of this section, several systems for which a risk analysis can be performed are presented.

I.2 Example of computer systems

The objective of this section is to present computer systems in which the risk of compromise by a malicious actor can be assessed. We call computer system any system composed of electronic elements that are interconnected to perform one or more tasks. An electronic element can consist of:

- Central Processing Unit (CPU) to perform arithmetic calculations and interact with external devices;
- short-term memory for quick access to data;
- long-term memory to save data;
- input and output devices to interact with the external environment.

The systems presented in this section provide a research context for the solutions studied in the literature review of this thesis. In the following part of this section, we will present and define three types of computer systems that are Information Technology (IT) systems, Cyber-Physical Systems (CPS) and embedded systems.

I.2.1 IT system

In this context, IT networks are composed of elements which collect, store, process, and distribute information. It includes technical elements such as servers, user workstations, and network components, but also human elements such as collaborators, administrators, and directors. IT networks exchange information through both wired and wireless networks. This type of system is used to support business processes of companies in a decentralized way, and can evolve over time to adapt to the growth of the company, by adding new elements and new communication technologies (cf Figure 4).

The objectives of the attacker in an IT system can be:

1. to make a service unavailable to impact the company's productivity;
2. recover credentials to access sensitive information;
3. to modify the content of a database.

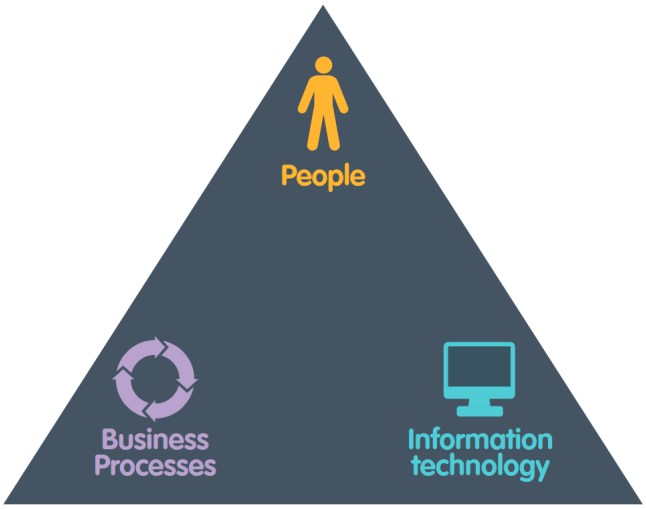


Figure 4: Composition of an information technology system [15]

I.2.2 CPS system

The role of CPS is to monitor and control several mechanized equipments such as robotic arms or electric relays. These systems are composed of a cyber part which allows to manage the communications between the elements of the network, to carry out calculations to make decisions according to the information coming from the environment and to store these data

to keep a history of the performed actions. They are also composed of a physical part such as sensors which allow to measure some properties of the environment of the system, as well as actuators to modify the state of the environment. These systems are used in the management of smart grid or in industrial production.

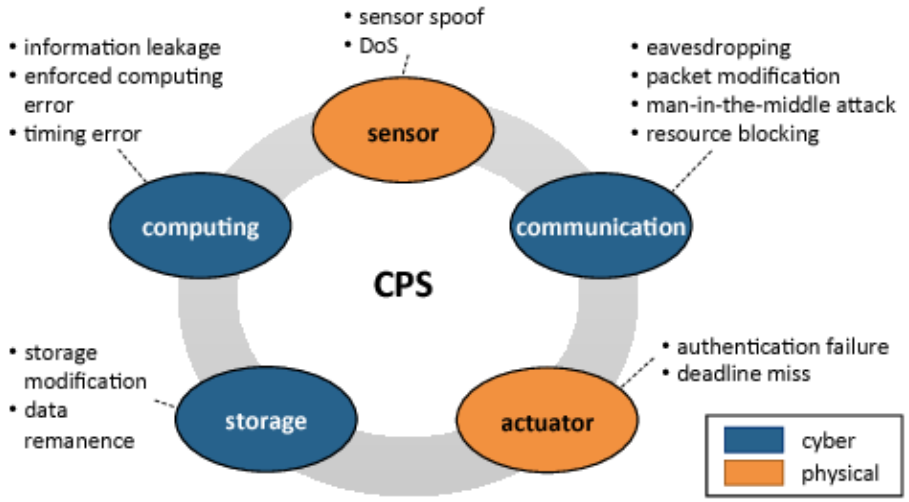


Figure 5: Various functions of a CPS [16]

The system responsible for the management of an industrial process is called an Industrial Control System (ICS). Signals are received from sensors that can for example measure temperature, pressure or distances in the environment. These signals are then transmitted to computer units called PLCs for Programmable Logic Controllers. The role of the PLC is to process the signals received on the input interfaces and to carry out calculations to decide on the signals to be sent on the output interfaces. These signals are then transmitted to actuators whose role is to modify the environment, such as activating a motor to move a robotic arm.

The attacker's objectives in a CPS can be:

- to make a PLC unavailable to degrade the productivity of an organization;
- retrieve information from technical documentation in order to steal intellectual properties;
- to corrupt the data sent to the actuators to create an industrial incident.

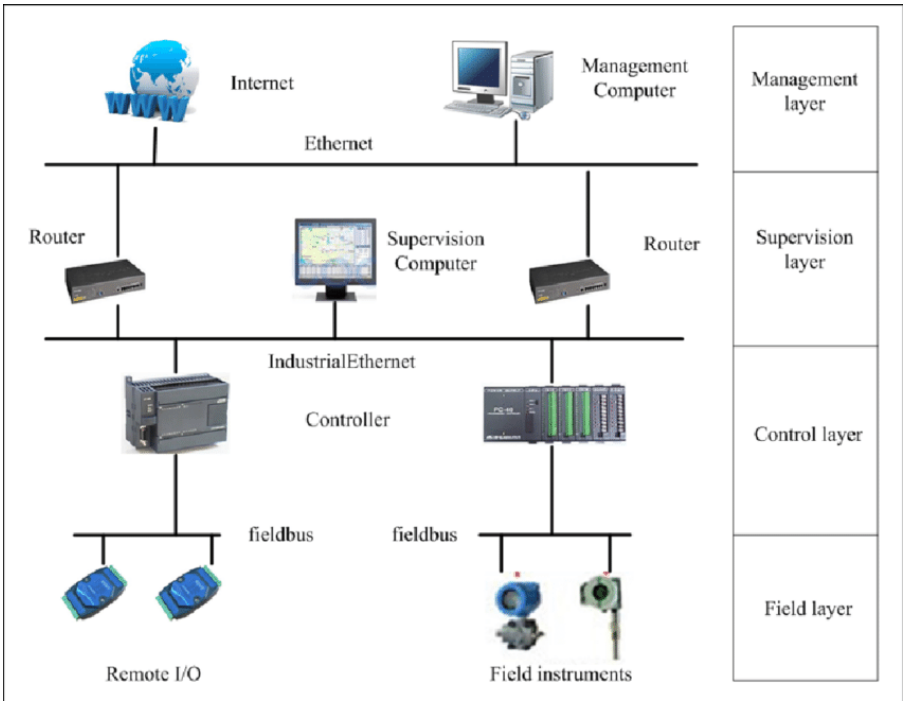


Figure 6: Composition of an industrial control system [17]

I.2.3 Embedded systems

Embedded systems are computer systems integrated into a larger physical system. These systems allow the control of devices that often have real-time computing constraints. An embedded system can for example be used for smart home or for controlling the autopilot of a vehicle. They have a similar role to the CPS in the sense that they interact with the environment through input-output interfaces, but with additional constraints concerning the electrical consumption which can be limited by the capacity and power of the installed batteries. The objectives of the attacker in an enbedded system can be:

- create a denial of service to negatively impact the user experience;
- recover personal information stored in the system;
- to corrupt the data sent by the sensors of an autonomous vehicle to create an incident.

The solutions identified in the literature review are intended to be used to assess the security of some or all of the systems previously presented. In the continuation of this section, risk analysis solutions based on multi-step attack modeling are presented.

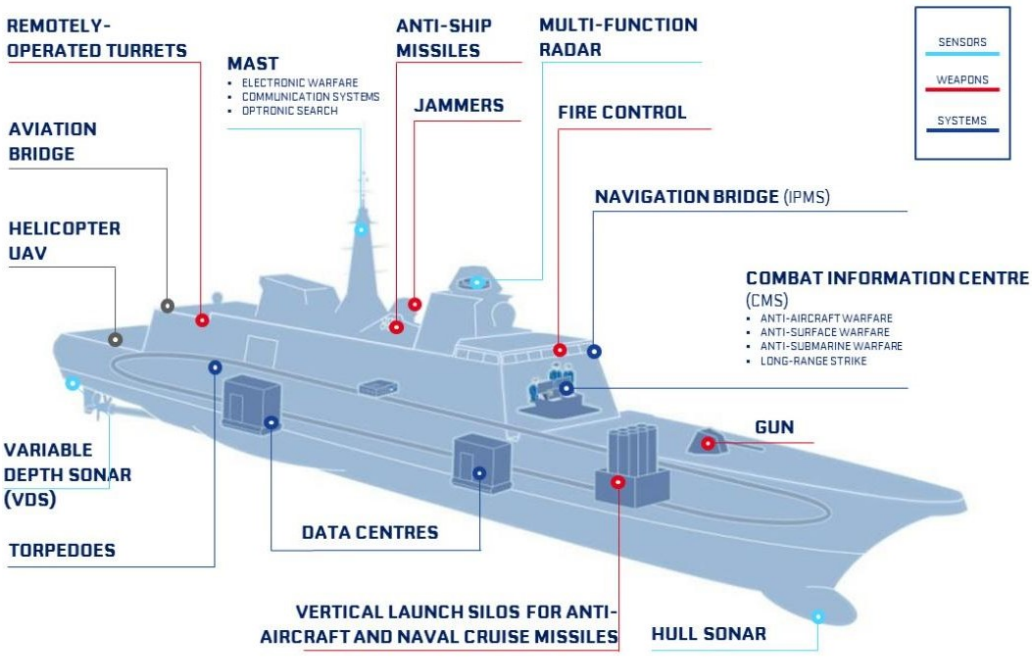


Figure 7: Example of an embedded system aboard a ship [18]

I.3 Multi-step attack modeling

I.3.1 Definition

When analyzing the security of a system, experts often have to answer the question: how can a malicious actor achieve their goals? The objectives of an attacker in a computer system are frequently summarized in the criteria of unavailability, integrity and confidentiality. For example, an objective may be to encrypt system data to make them unavailable, to modify the input values of a PLC to impact the industrial process or to steal confidential information about a new technology. To identify the different ways in which an attacker can achieve its objectives, it is possible to list all the actions that must be performed. The successive realizations of these actions form what we call attack paths. It is possible to model these attack paths as a tree where the root node represents the attacker's objective. The steps to reach this goal

are represented with child nodes. The relationship between nodes can be conjunctive when all child nodes must be realized to reach the parent node or disjunctive when several actions are possible to reach the parent node.

I.3.2 Attack graph generation

Several articles propose methods to build attack trees and use them to analyze the security of computer systems [19]–[23]. Unlike attack trees, attack graphs allow to represent the attack paths in a more compact way by avoiding repeating some nodes.

The first attack graphs were made of nodes representing the different states of the system and arcs modeling the actions of the attacker allowing to go from one state to another. These graphs can be generated automatically from a description of the system with model checking algorithms such as SMV [24] or NuSMV [25]. The problem with these attack graphs is that their size grows exponentially with respect to the size of the system. It is to solve this problem that the monotonicity assumption has been proposed in [26]. This assumption says that one action of the attacker cannot invalidate a precondition of another action. However, it is not always true. For example, when an attacker exploits a vulnerability on a network service and causes its unavailability, it will no longer be possible to exploit other vulnerabilities on this service. The attack graph models presented in the rest of this chapter all use monotonicity assumption to reduce the size of the graph.

The network visible in Figure 8 was used to test the different attack graph generation methods. This network comes from [27] and it is composed of a web server, an authentication server on a DMZ network, and a database server on an internal network. Several vulnerabilities are present in this system and are summarized in Table 2. The web server hosts an Apache web service that is vulnerable to an attack that allows the execution of arbitrary code. An SSH service is present on the authentication server. The vulnerability CVE-2002-0640 is present on this service and allows the attacker to execute arbitrary code on the server. A MySQL service is present on the database server. This service is vulnerable to an attack allowing to execute arbitrary code on the server. A second vulnerability is present on the Linux kernel of the database server. This vulnerability is locally exploitable and allows the attacker to elevate his privileges on the server.

The tool TVA (Topological Vulnerability Analysis) [28], [29] automatically builds an attack graph from attack scenarios, a set of vulnerabilities identified in the system and network modeling. The attack scenarios are made up of the attacker's objectives and initial conditions. The

Software	Vulnerability (CVE-ID)	Short symbol
Apache Web Server v1.3	CVE-2006-3747	V1
OpenSSH v2.3.1-v3.3	CVE-2002-0640	V2
MySQL v4.0, v5.0	CVE-2009-2446	V3
Linux Kernel v2.4, v2.6	CVE-2004-0495	V4

Table 2: List of vulnerabilities present in the network

list of vulnerabilities can be obtained automatically with scanning tools such as Nessus [11] or Retina IoT (RIoT) [30]. The attack graph is composed of nodes representing the vulnerabilities and security conditions acquired by the attacker. The arcs from the precondition nodes to a vulnerability node represent the conditions required to exploit the vulnerability. An arc from a vulnerability node to a condition node represents the impact of the vulnerability exploitation. The size of the attack graph evolves in a quadratic way with respect to the size of the system. Figure 9 shows the attack graph generated from the test network using the TVA method.

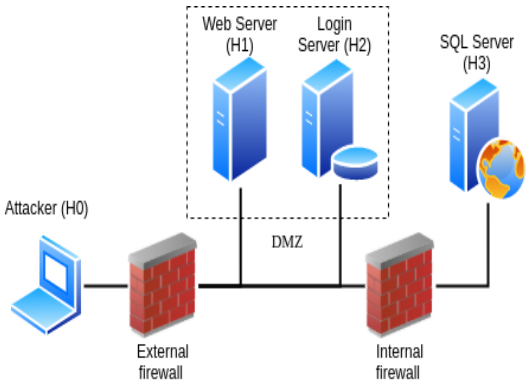


Figure 8: Configuration of the network used as an example [27]

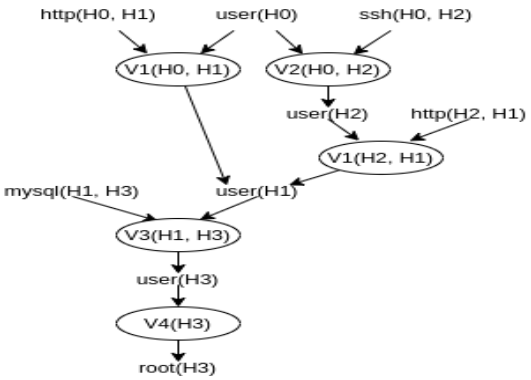


Figure 9: Exploitation dependency type attack graph used in TVA [27]

The MulVAL [31] (Multihost, multistage Vulnerability Analysis) framework automatically generates an attack graph from a list of vulnerabilities present in the system, asset configuration, user access rights, possible interactions and policies. The system is modeled with the Datalog language [32]. The possible interactions in the system are modeled with Horn clauses. A Horn clause is a logical formula written as a rule. They allow to model reasoning rules and to deduce new information about the system from already known facts. The XSB [33] environment is used to deduce new information about the system state. Figure 10 shows the attack graph generated from the test network using the MulVAL framework. The structure of the attack graph is relatively identical to that of TVA with nodes representing vulnerability exploits, parent nodes representing the necessary preconditions and a child node representing the impact of the attack. The attack graph construction algorithm has a worst-case time complexity $O(n^2 \times log(n))$, with n the number of hosts in the system and its size evolves in a quadratic way [34].

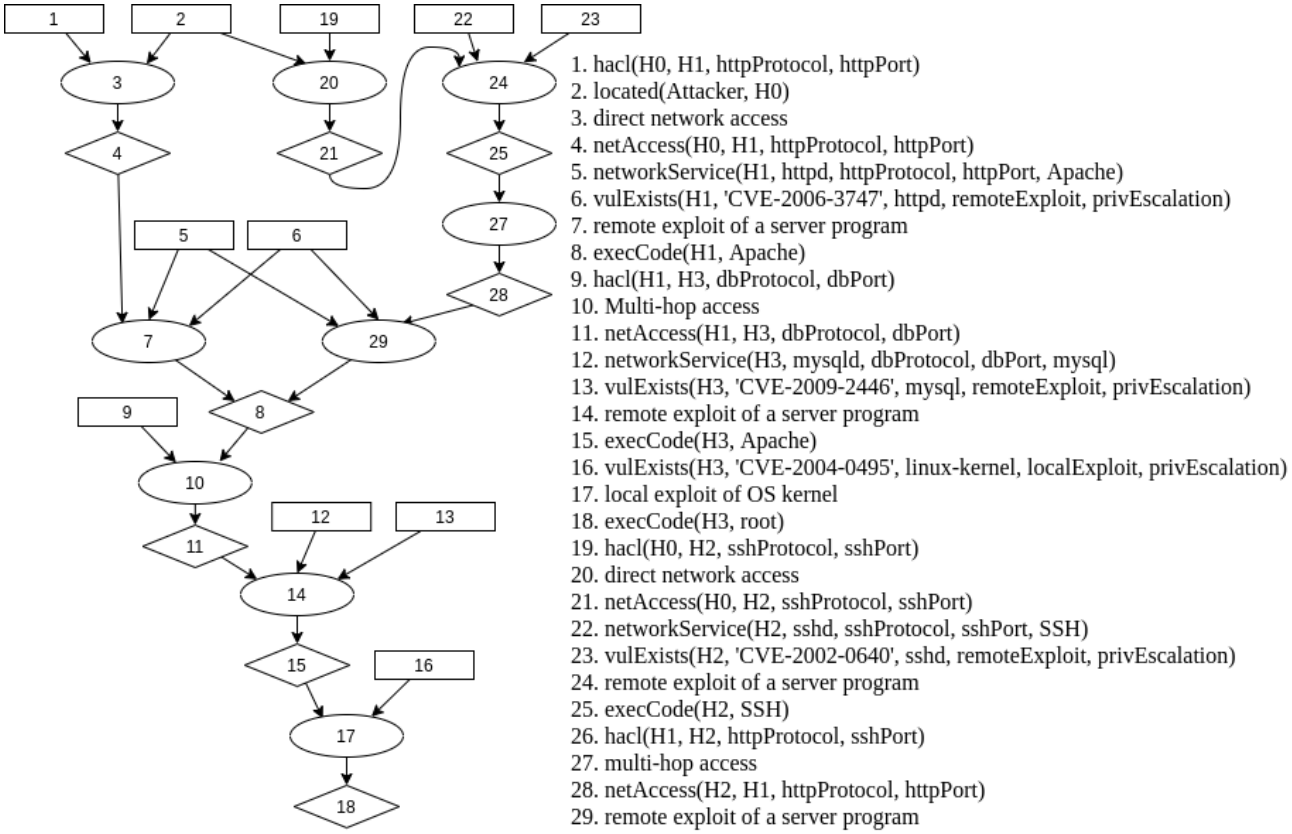


Figure 10: Logic-based attack graph used in MulVAL [27]

The NetSPA (Network Security Planning Architecture) [35] framework allows to automatically build an attack graph called multiple-prerequisite graph. First, a reachability graph is built from information about the network topology, network filtering equipment and hosts. The reachability matrix is simplified into a matrix of reachability groups by removing any redundancies. Vulnerabilities can be identified automatically with a scanning tool. Then, using information from databases such as NVD, the vulnerabilities discovered on the network are analyzed to deduce whether they are remotely or locally exploitable, and what is its impact on the host. The impact of an attack on a host is measured with one of the following criteria: root, user, DoS or other. In this article, a vulnerability is defined as any method by which an attacker can gain access to the system. However, attacks such as social engineering and physical attacks are not modeled because they cannot be identified automatically. Figure 11 shows the attack graph generated from the test network using the NetSPA framework. The attack graph is composed of 3 types of nodes: state, prerequisite and vulnerability nodes. The state nodes represent an attacker's state in the system and allow unlocking new prerequisites. The prerequisite nodes represent the conditions required to exploit a vulnerability. Vulnerability nodes represent the vulnerabilities present on the hosts of the system and whose exploitation

allows the attacker to reach a new state. This attack graph structure allows to avoid repetitions of some nodes as it can happen in TVA. The performance is similar to MulVAL. The worst-case time complexity of the attack graph generation algorithm is $O(max(V,T) \times R \times C)$, with V the number of vulnerabilities, T the number of network ports, R the number of reachability groups and C the number of credentials.

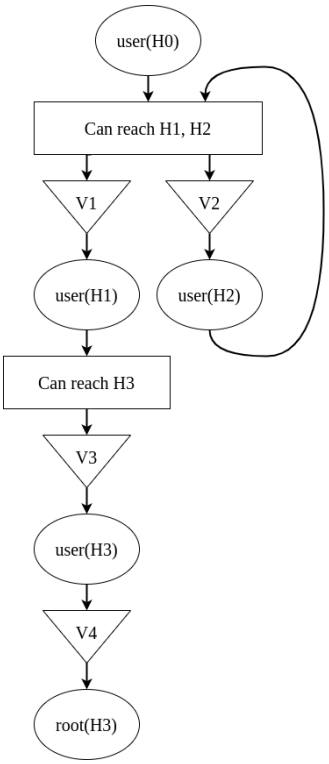


Figure 11: Multiple prerequisite attack graph used in NetSPA [27]

Although the attack graphs make it possible to identify the attack paths leading to the compromise of the system components, the difficulty of the actions that the attacker must perform is not considered. We will introduce some methods that have been proposed to assess the attacker's chances of success from the attack graphs.

I.3.3 Attack graph-based risk assessment

Attack graphs can be used to assess the risk of system compromise by considering the difficulty of the attacker's actions. It is possible to list all the attack paths present in an attack graph and calculate for each of them their probability of success as in [36]–[41]. In [36], the authors aim to reduce the cyber risk to the Internet of Things (IoT) as effectively as possible, considering the budgetary limitations that are imposed. A backward algorithm is

used to traverse the attack graph and list all attack paths. This algorithm ignores paths that do not lead to any of the attacker's objectives. Paths that do not start at the attacker's initial condition node are also removed. If an exploit node is traversed a second time, path extraction is stopped to avoid cycling. Attack paths that exceed a given length or have a probability of success below a given limit are removed. The success probability of an exploit E_i is calculated using the CVSS score normalised from the following equation:

$$Eprob(E_i) = (8.22 \times AV \times AC \times PR \times UI)/10 \tag{5}$$

where AV is the attack vector, AC is the complexity of the attack, PR is the required privileges, and UI is the user interaction. Exploiting the same type of vulnerability several times in the same attack path increases its probability of success. This characteristic is considered with the parameter $\theta \in [0;1]$, and allows to model the learning of the attacker. If $\theta = 0$, the probability of exploitation of the same type of vulnerability does not change. With $\theta = 1$, the probability of exploiting the same type of previously exploited vulnerability is 1. The probability of successful exploitation of an attack path is calculated by the following equation:

$$Aprob(A_j) = \prod min(1.0, Eprob(E_i) + \theta \times \varphi \times (1 - Eprob(E_i))) \tag{6}$$

where φ is the number of exploits of the same type as E_i. The Security Metric noted SM allows to measure the security of the system and the attack surface, and is calculated with the following equation:

$$SM = \sum Aprob(A_j) \tag{7}$$

The solution was tested on an attack graph composed of 25 nodes. Nine attack paths are found with the back propagation algorithm. Among them, four are removed, because their starting node is not the attacker node, and the last one, because the number of nodes in the path exceeds the allowed limit. The time complexity of the attack path extraction phase is $O(M^{n-1})$, where M is the maximum number of exploits that point to a condition, and n is the maximum length of the attack paths that has been defined. Experimental tests have been performed with different attack graphs of increasing complexity (see Table 3).

Attack graph	Number of initial conditions	Number of exploits	Number of arcs
A	21	32	129
B	98	305	1382
C	512	984	4982
D	945	1326	6814
E	2122	4023	21221

Table 3: Characteristics of the different attack graphs used for performance evaluation.

The arcs are randomly generated and the minimum probability of an attack path is set to 0.01. 50 simulations are performed for each network and the average of the results is calculated. Figure 12 shows the average number of attack paths identified as a function of the complexity of the different attack graphs and the value n representing the maximum length of an attack path. Figure 13 shows the average time required for the identification of the attack paths in the different attack graphs and for different values of n.

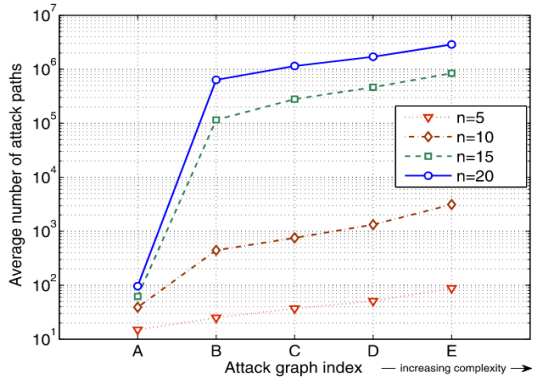


Figure 12: Average number of attack paths for different randomly generated attack graphs with different path length limits n [42]

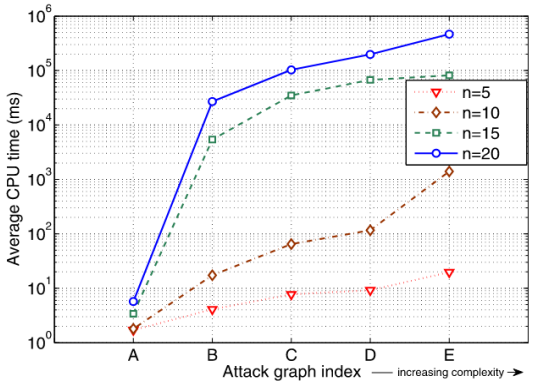


Figure 13: Average CPU time in milliseconds for the attack path generation phase for different randomly generated attack graphs with different path length limits n [42]

We notice that the given complexity $O(M^{n-1})$ is equal to $O(M^{M-1})$ when no attack path length limit is set, which means that the complexity is exponential in this case. With a value of $n = 5$, the complexity becomes $O(M^4)$. The time complexity of the attack path identification algorithm is therefore highly dependent on the value n that is set.

The experimental analysis of the complexity is difficult to interpret. The x-axis corresponds to the 5 attack graphs of increasing complexity (see Table 3), but the notion of complexity has not been clearly defined. It corresponds to a combination of the number of initial condition nodes, the number of exploitation nodes and the average number of arcs. Moreover, these various parameters do not evolve linearly. For example graph A has 32 exploitation nodes, 305

for graph B and 984 for graph C. This is an increase of factor 9.5 between graph A and B, and of factor 3.2 between graph B and C.

In [41], the authors aim to develop a method for assessing the cyber risk of port infrastructures using attack graphs. These infrastructures are dependent on complex supply chains that rely on IT networks, involving a large number of organisations such as transport, energy and telecommunications. Network assets may be vulnerable, allowing an attacker to exploit them to compromise confidentiality, integrity or availability. All the attack paths present in the graph are listed using a deep traversal algorithm. A set of constraints is used to reduce the complexity of the algorithm:

- the attacker's capabilities whose possible values are shown in Table 4;
- the position of the attacker, whose possible values are shown in Table 5;
- the maximum propagation length;
- the entry and exit points of the attack.

Qualitative values	Description
High	The attacker is an expert and has sufficient resources to carry out the attack
Medium	The attacker's expertise and resources are of moderate level
Low	The attacker has limited expertise and resources to carry out the attack

Table 4: Attacker's capabilities

Qualitative values	Description
Local	The attacker is located on the local network
Adjacent	The attacker is located on a adjacent network
Network	The attacker is located on a remote network such as the Internet

Table 5: Attacker's location

The use of entry and exit points makes it possible to carry out the risk analysis on a sub-part of the system, which makes it possible to constantly re-evaluate the cyber risk in a context of high network evolution.

The model was tested in two different cases. The first is a simplified example consisting of 7 assets and 9 vulnerabilities and was used to test the effectiveness of the algorithm. Three tests with different parameters were performed and all the attack paths were identified (see Table 6).

Test number	Number of existing paths	Number of identified paths	Paths
1	2	2	[1, 2], [1, 5]
2	0	0	∅
3	1	1	[3, 4, 5, 7]

Table 6: Results of the analysis of the first case study

The second use case is inspired by the port of Valencia and consists of 182 assets. The performance of the solution was tested and the results are available in Figure 14. The y-axis corresponds to the execution time of the algorithm in seconds. The terms low, medium and high correspond to the different capacity levels of the attacker. The values 5, 10, 15 and 20 correspond to the total number of entry points and targets. The attacker's position was set to local. The maximum length of the propagation in the network is fixed at 10.

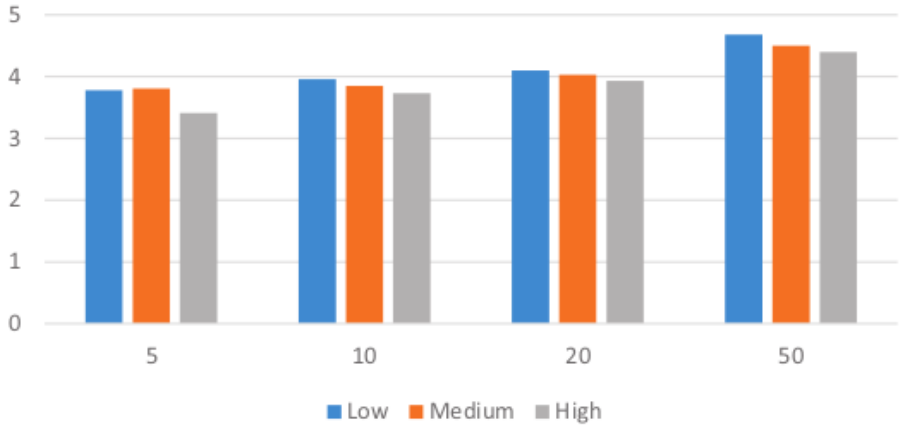


Figure 14: Performance evaluation of Algorithm 1 in the use case of the Valencia port [42]

The benchmark that has been carried out does not show an explosion of computing time in relation to the number of entry points and objectives. This is certainly due to the many parameters that eliminate some of the attack paths considered to be too difficult for the attacker to exploit. However, it would be interesting to measure the number of false negatives. The complexity of the algorithm has not been given. The algorithm presented in this paper is very similar to the one proposed in [36]. The main differences is that the probability of the attack paths is not considered in this article, but additional parameters have been defined such as entry and exit points, as well as the capabilities and position of the attacker.

Other solutions have been proposed to compute the probabilities for each node of the attack graph without having to list all the attack paths as in [43]–[47]. In [43], the authors propose a solution to assess the security of IT systems. This assessment is automatically performed to not rely on human work that could be error-prone. The authors propose a new modeling

language similar to the Meta Object Facility (MOF) [48], where the system is represented in several layers with an object-oriented structure. This language allows a reduction in the cost of modeling the system, a great flexibility when defining new components and a possible reuse of the models.

The layer 0 of the model visible in Figure 15 allows to define the construction rules of the attack graph. It is composed of the following elements:

1. the class *asset* represents the system components.
2. the relationships between components are represented in class *AssetRelationship*, such as a physical connection between two components.
3. class *AttackStep* represents actions that allow an attacker to compromise a component. The time required to achieve an attack step is represented by a probability distribution over the Time To Compromise $\boldsymbol{TTC}$. This probability distribution makes it possible to model the uncertainty about the value of the time required to carry out the attack. There are two types of attack step:
   - the as_{min} attack step can be carried out by the attacker if at least 1 parent attack step has been achieved;
   - the as_{max} attack step can be carried out by the attacker if all the parent attack steps have been achieved.
4. class *Attacker* is used to model the entry points of the attack with attack steps having a $\boldsymbol{TTC}$ value of 0.
5. class *AttackStepRelationship* represents the relations between the attack steps and models the possible propagation of an attacker.

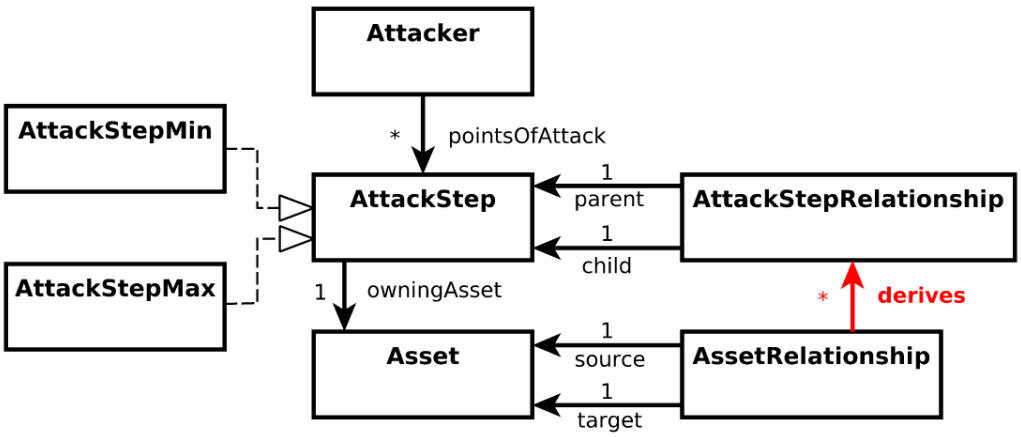


Figure 15: Layer 0 of the system modeling [42]

The attack graph G constructed from this model is composed of:

- a set V of nodes representing the attack steps;
- a subset $I \subset V$ of nodes representing the entry points of the attack;
- a set $E \subseteq V \times V$ of arcs between the nodes representing the possible progressions of the attacker;
- a weight attribution function $w : (A,B) \in E \rightarrow P(TTC_A)$ which allows to define the probability distributions of the time required to perform an attack step by the attacker.

The layer 1 of the model visible in Figure 16 allows to represent the different threats that exist in the system. All classes of this layer are instances of the class Asset of layer 0. Each class of layer 1 contains a set of attack steps that are related to this class. The attack steps framed in dotted lines represent as_{min} and those framed in lines represent as_{max}. An arc between two classes represents a relationship that is part of class *AssetRelationship* of layer 0. A dotted arc between two attack steps represents a possible progression of the attacker and is part of the class *AttackStepRelationship* of layer 0. The four classes defined in layer 1 are:

- identity that represents an authorization required in the system to perform an action;
- agent that represents an entity in the network such as a person, a software or a physical component;
- data that represents information such as a file, credentials or a command;

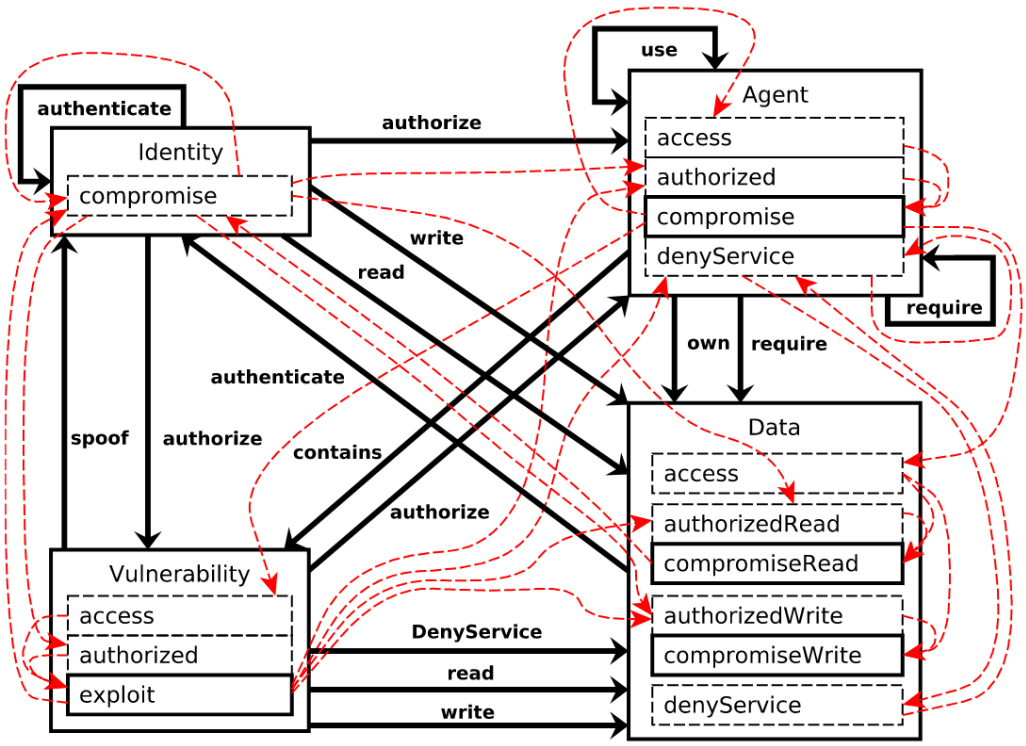


Figure 16: Layer 1 of the system modeling [42]

- vulnerability that represents a flaw in the implementation or design of the system. The uncertainty about the actual existence of the vulnerability is represented by a probability distribution and is considered in the computation of the TTC value.

It is possible to represent some processes in more detail by adding a new modeling layer. For example, Figure 17 represents the authentication process of a user, where class *Host1* and *Host2* are instances of class *Agent*, classes *User* and *Admin* are instances of class *Identity*, and class *UserLoginCredentials* an instance of class *Data*.

The calculation of the TTC value for each node of the attack graph is performed in two steps:

1. a TTC value is sampled from the probability distribution that represents the time required to complete the action, given that all the parent attack steps have been completed;
2. the minimum TTC value that considers the parent nodes is computed for each node of the attack graph using a modified version of Dijkstra's shortest path algorithm [49].

This process is performed multiple times and allows to obtain the distribution of the success

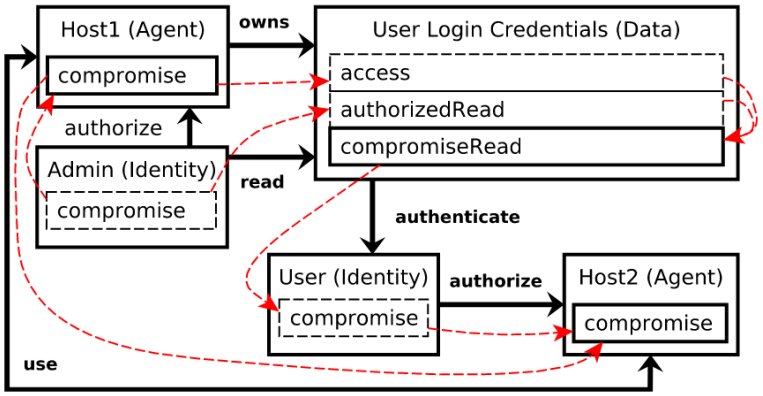


Figure 17: Representation of the authentication process in a third modeling layer [42]

frequencies of an attack step as a function of time.

But there is not enough information about how the attack graph is generated from the system modeling, as well as how the initial TTC value of a vulnerability is computed from the probability of its actual existence. There is also no analysis of the complexity of the algorithms used such as the modified version of Dijkstra's shortest path algorithm. The model has not been tested on a large system, so it is not certain that this solution is able to assess them.

In [46], the authors observe that identified vulnerabilities may not be immediately patched due to environmental factors, cost or the organisation's mission. Their objective is to propose a solution to identify the vulnerabilities that need to be addressed as a priority, by computing the probability that the attacker will succeed in compromising some system components.

An attack graph G is defined as $G(E \cup C, R_r \cup R_i)$, where E is the set of exploit nodes, C the set of condition nodes, $R_r \subseteq C \times E$ the set of incoming arcs to an exploit node and $R_i \subseteq E \times C$ the set of incoming arcs to a condition node. The individual score of a vulnerability corresponds to the probability of successful exploitation by the attacker assuming that all required preconditions are satisfied. The cumulative score combines the individual score of the vulnerability with the structure of the attack graph. The authors use an attack graph to model the dependencies between vulnerability exploits, represented by ovals, and the security conditions acquired by the attacker, represented by text.

The calculations of the attacker's success probabilities that will be presented do not consider the conditional interdependencies between attacks. The cumulative score $P(e)$ of an exploitation type node is computed from its individual score $p(e)$ and the cumulative score of the preconditions $P(c)$ with the following equation:

$$P(e) = p(e) \times \prod_{c \in R_r(e)} P(c) \tag{8}$$

The cumulative score $P(c)$ of a condition node is computed from its individual score $p(c)$ and the cumulative score of the vulnerabilities $P(e)$ allowing to reach the condition c, with the following equation:

$$\begin{aligned} P(c) &= \begin{cases} p(c),\ R_i(c) = \emptyset \\ p(c) \times \oplus_{e \in R_i(c)} P(e),\ otherwise. \end{cases} \\ &\text{with } \oplus \text{ recursively defined as } \oplus P(e) = P(e), e \in E \\ &\text{and } \oplus (S_1 \cup S_2) = \oplus S_1 + \oplus S_2 - \oplus S_1 \times \oplus S_2, S_1 \subseteq E \text{ and } S_2 \subseteq E \end{aligned} \tag{9}$$

The authors address the problem of cycles in attack graphs. Some cycles can be deleted (left-hand graph in Figure 18) when a condition within the cycle can never be satisfied. Others can be broken (middle graph of Figure 18) when an exploit within the cycle only allows to obtain a condition which is itself necessary for the exploit. However, some cycles can neither be deleted nor broken (right-hand graph in Figure 18).

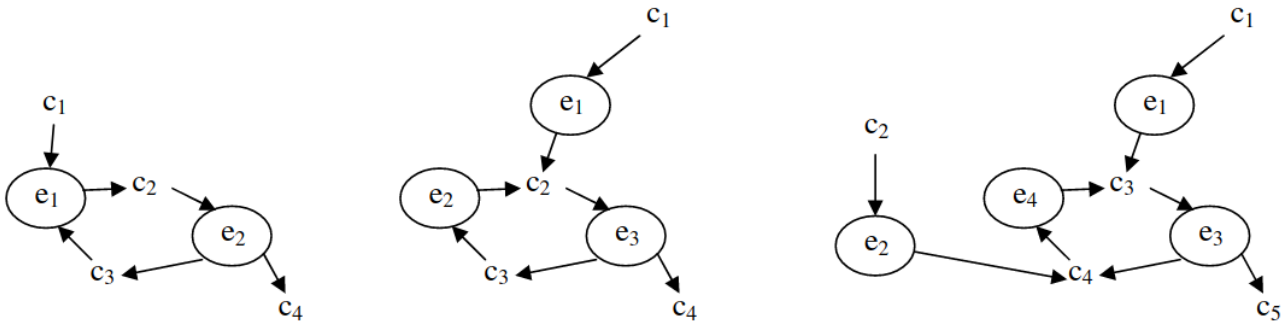


Figure 18: Possible cycles in an attack graph [42]

The cumulative score cannot be computed for a node present in a cycle. To solve this problem, the authors propose to generate a graph $A(G,v)$ from the attack graph G where all outgoing arcs from node v are deleted, and recursively all arcs and nodes that are no longer accessible. This allows the node v to be no longer present in a cycle and to compute its cumulative score with equations (8) and (9) in the graph $A(G,v)$. The authors prove that computing the cumulative score of a node v in graph $A(G,v)$ is not a problem because only nodes that have node v as a predecessor are deleted and thus it has no impact on the attack

paths reaching node v.

In order to compute the cumulative score of all nodes in the attack graph, the graph is traversed with a modified Breadth-First Search (BFS) algorithm, where a node v can only be computed when all cumulative scores of the parent nodes have already been computed. The graph traversal with the BFS algorithm stops when there are only nodes that are entry points in cycles. At this moment, all the cumulative scores of these nodes are computed by generating the graph $A(G, v)$. Then the BFS algorithm is used again and so on until the entire attack graph has been traversed.

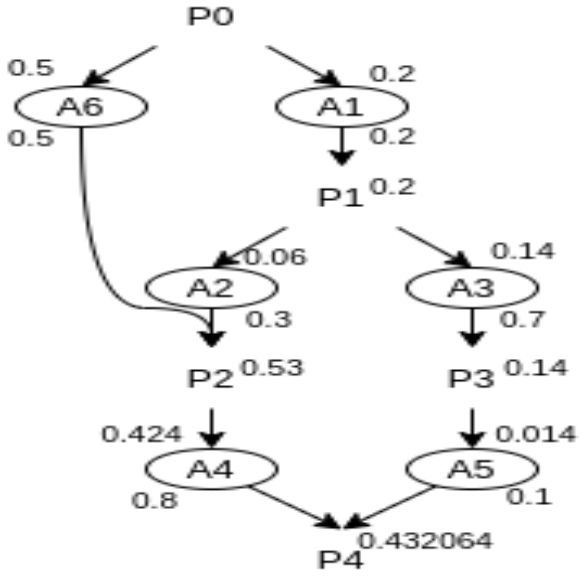


Figure 19: Attack Graph used to check the solution proposed in [46]

This solution has the advantage of not listing all the attack paths present in the attack graph, their number evolving exponentially with the size of the graph, which makes its use possible in large systems. However, the authors do not consider the conditional interdependencies between nodes. Yet it is obvious that in Figure 19, action $A5$ depends on condition $P1$, and that action $A4$ also depends on condition $P1$. It is therefore necessary to consider this shared dependence when computing the probability $P(P4)$. Attack graph in Figure 19 is used to demonstrate that this results in incorrect probabilities. The probabilities below the nodes correspond to the individual scores, while the probabilities above correspond to the cumulative scores computed with the solution proposed in this article. We constructed the probability tree corresponding to the attacker reaching condition P_4 (see Figure 20), which allowed us to compute the probability $P(P_4)$. We obtain $P(P_4) = 0.43072$, which is a different result than by following the solution proposed in this article, showing the importance of considering the

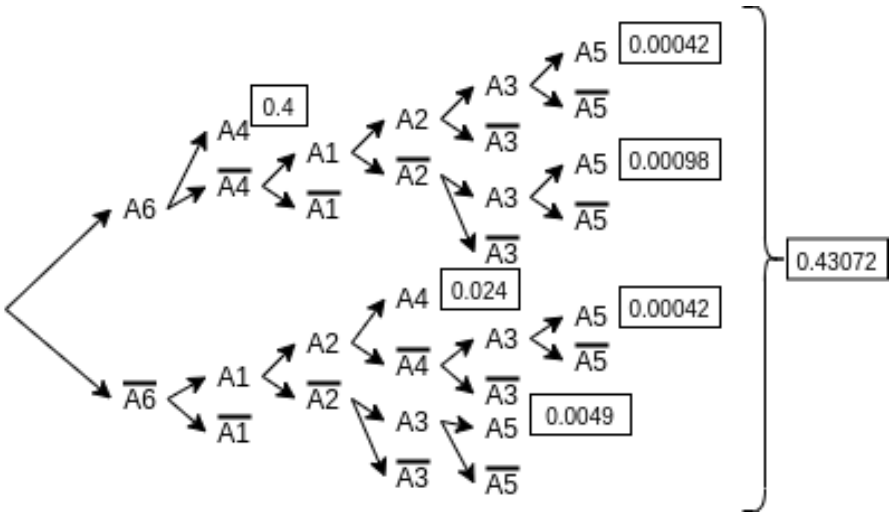


Figure 20: Probability tree allowing to reach P_4

dependencies between vulnerability exploits.

In [47], the authors' objective is to compute the probability that the attacker will reach some given security conditions in the system based on the attack paths present in an attack graph. The MulVAL [31] framework is used to build the attack graph. There are three kinds of nodes in the graph:

1. attack-step nodes G_C representing a logical AND relationship with its parent nodes of type privilege;
2. privileges nodes G_D representing a logical OR relationship with its parent nodes of type attack-step;
3. configuration nodes representing the initial properties of the system.

Configuration nodes are removed and substituted by a single root node $G_R \in G_D$, with an arc to all attack-step nodes that are left without parents. Privilege nodes with several outgoing arcs are called branch nodes, $G_B \in G_D$. The set of probabilities G_M associated with the attack-step nodes can be computed from the CVSS score. For an attack-step node $c \in G_C$, let $G_M[C]$ be the probability associated with node c such that $Pr[e|P] = G_M[C]$. The probability G_V of an attack being carried out by the attacker is associated with the root node G_R, with $G_V = 1$ in this article.

The concept of d-separation is used to establish conditional interdependencies between nodes and it has been adapted to the specific case of attack graphs. A set $V \subseteq G_B$ is said to d-separate

distinct sets of nodes A and B if among all diverging paths between A and B, there is a node $v \in V$ such that v is the divergence point. Theorems (1) and (2) are used to compute the marginal probability $Pr[N]$.

Theorem 1 $\forall D, N \subseteq G_N, Pr[N] = \sum_D Pr[N|D] \times Pr[D]$

Theorem 2 $Pr[N|D] = \prod_{n \in N} Pr[n|D]$, *with D the node set that d-separate any pair of nodes in the node set N.*

To compute the probability of a node v present in a cycle, the authors start by listing all acyclic paths that pass through this cycle. Then, only the paths passing through node v are kept. The probability that the node v is true is equal to the probability that at least one of the attack paths passing through this node is true. The calculation of this probability is done in the same way as for a node that is not present in a cycle with theorems (1) and (2).

The solution has been tested in different systems, with generated attack graphs having a different number of cycles. The platform used to perform these tests is not specified, but what interests us here is to measure the way the computation time evolves as a function of the system size. The tests carried out show that the number of cycles present in the attack graph has a great influence on the calculation time. For example, in a system composed of 9 hosts, it takes 4 seconds to perform the computations in an attack graph with 28 nodes in the largest cycle, against 8 minutes and 31 seconds with 46 nodes in the largest cycle. Although this solution makes it possible to compute compromise probabilities by considering conditional dependencies between vulnerability exploits, execution times increase quickly, especially when many cycles are present in the attack graph. Therefore, it seems that this solution is not suitable for large systems, where many cycles may be present in the attack graph. This solution was tested on the attack graph shown in Figure 19 and the probability $P(P4)$ corresponds well to the value obtained from the probability tree shown in Figure 20.

I.3.4 Bayesian attack graph

A Bayesian network is an acyclic graph where the nodes represent random variables and the arcs represent dependencies between them. Each node is associated with a Conditional Probability Table (CPT) where each row represents the probability of the random variable X_i

regarding the value of its parents, noted $P(X_i|PARENTS(Xi))$. Once all the probability tables have been completed, it is possible to compute the joint probabilities of a set of random variables X_i with equation (10). Then, the marginal probabilities of a random variable X_i is calculated with equation (11).

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|PARENTS(X_i)) \tag{10}$$

$$P(X_i = x) = \sum_{X_1} ... \sum_{X_{i-1}} \sum_{X_{i+1}} ... \sum_{X_n} P(X1, ..., X_i = x, ..., X_n) \tag{11}$$

It is also possible to compute the marginal probabilities of the random variables X_i given one or more evidences E with equation (12). An evidence is a random variable whose value is known, such that $E_i = e_i$. The hidden variables Y are those whose value is not known.

$$\begin{aligned} &P(X_i = x|E = e) = \frac{P(X_i = x, E = e)}{P(E = e)}, \text{ with } P(X_i = x, E = e) = \sum_{Y} P(X_i = x, Y, E = e) \\ &\text{and } P(E = e) = \sum_{X} \sum_{Y} P(X, Y, E = e) \end{aligned} \tag{12}$$

The concept of d-separation makes it possible to obtain the set of random variables on which it is necessary to marginalize to compute the probability $P(N)$. The marginalization computation of random variables is generally NP-hard [50]. It can be done in two ways:

1. exact algorithms such as junction tree and cut-set conditionning [51]. In the case where the graph is a polytree, there is Pearl's message passing algorithm [52] of polynomial complexity. As a reminder, a polytree is a directed acyclic graph for which there are no undirected cycles. Another solution is to use a large number of CPU cores with the junction tree algorithm as in [53]. This allows, for the computation of the marginalization of a node with two states and 30 parents, to go from $19,463$ seconds ($\approx 5h$) with 2 CPU cores, to 55 seconds with 1024 CPU cores.
2. In an approximate way by carrying out sampling as in [54]. The simplest method is called Rejection Sampling: it consists in sampling random variables by respecting the conditional probabilities of the Bayesian network, and by removing the samples which do not correspond to the observed random variables. Then the value of the marginal

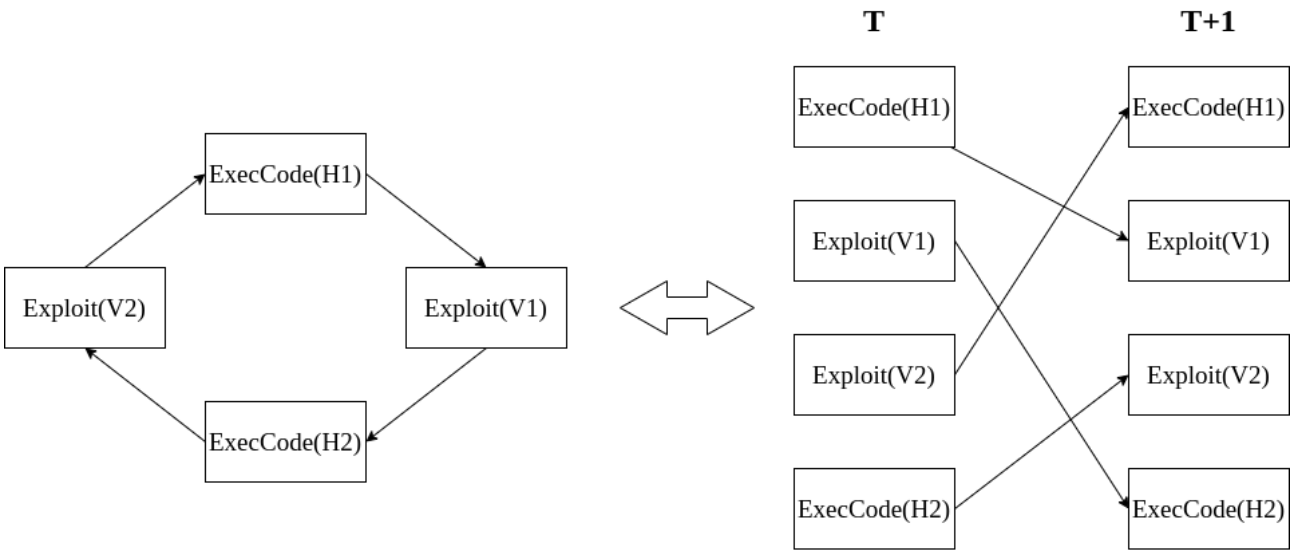


Figure 21: Removing cycles in a Bayesian network using a dynamic Bayesian network

probabilities of each node can be determined by counting the number of corresponding samples divided by the total number of samples. Other approximate algorithms exist: in paper [55], the authors measured the performance of Likelihood Weighting [56], Gibbs algorithms [57], and Pearl inference [52]. A probability is calculated with the juntion tree exact algorithm and is used as a reference to evaluate the performance of the approximate algorithms. The Gibbs algorithm starts to converge to the correct value after 400 iterations and gets very close to it after 600 iterations. The likelihood weighting algorithm is directly very close to the correct value and gets very close to it after 600 iterations as well. The Pearl inference algorithm is the fastest to converge to the correct value and gets very close to it in only 3 iterations. However, the computation time in relation to the number of iterations increases more rapidly than the Gibbs algorithm, but less than the likelihood weighting algorithm. If we take into account the computation time, the Gibbs algorithm is the fastest to converge to the correct value.

Dynamic Bayesian networks are an extension of static ones. They are a series of static Bayesian networks linked by temporal arcs. This allows the impact of time in the system to be considered. It also makes it possible to remove cycles that could appear in a classic Bayesian network. Figure 21 shows an example of cycle removal using a dynamic Bayesian network.

Bayesian networks can be used to model conditional interdependencies between nodes in an attack graph and compute the attacker's success probabilities as in [58]–[64]. These types of networks are called Bayesian attack graphs. There are two surveys concerning the use

of Bayesian networks in attack graphs [65], [66]. In [58], the objective is to propose a risk management framework based on Bayesian networks allowing to compute the probability of network compromise. A Bayesian attack graph is used to model the causal relationships between the different states of the network. The probability of successful exploitation are computed based on the version 2 of the CVSS score of the associated vulnerability with equation (13).

$$Pr(e_i) = 2 \times AccessVector \times AccessComplexity \times Authentication \tag{13}$$

Several concepts are defined:

- a template is a generic network property such as a vulnerability, account access, system or network properties. For example, a template could be a buffer overflow vulnerability on the SSH service of an FTP server;
- an attribute S defines the truthfulness of a template, with $S = True$ or $S = False$. We note $Pr(S)$ the probability that the attribute S is true, and $Pr(\neg S)$ that it is false;
- dependencies between two attributes are defined by the set A such that $A : S \times S \rightarrow [0,1]$. An atomic attack is defined as a relation $a : S_{pre} \rightarrow S_{post}$, with S_{pre} the precondition necessary for the exploitation to be carried out, and S_{post} the postcondition of the exploitation, such that:
  - $S_{pre} \neq S_{post}$;
  - If $S_{pre} = 1$, then $S_{post} = 1$ with a probability $A(S_{pre}, S_{post}) > 0$;
  - $\nexists S_1, ..., S_j \in S - \{S_{pre}, S_{post}\}$ such that $A(S_{pre}, S_1) > 0$, $A(S_1, S_{post}) > 0$, ..., $A(S_{pre}, S_j) > 0$, $A(S_j, S_{post}) > 0$.

  The probability of success of an atomic attack is given by $Pr(e_i)$, such that $A(S_{pre}, S_{post}) = Pr(e_i)$;
- a Bayesian attack graph is defined by the set $BAG = (S, \tau, \varepsilon, P)$ with:
  - $S = N_{internal} \cup N_{external} \cup N_{terminal}$, with $N_{internal}$ the set S_i for which $\nexists a \in A | S_i = post(a)$, $N_{internal}$ the set S_j for which $\exists a_1, a_2 \in A | (S_j = pre(a_1), S_j = post(a_2))$, and $N_{terminal}$ the set S_k for which $\nexists a \in A | S_k = pre(a)$;
  - $\tau \subseteq S \times S$, $(S_{pre}, S_{post}) \in \tau$ if $S_{pre} \rightarrow S_{post} \in A$;
  - ε is a set of relations (S_j, d_j) defined for all $S_j \in N_{internal} \cup N_{terminal}$ and $d_j \in \{AND, OR\}$. $d_j = AND$ if $S_j = 1 \implies \forall S_i \in Pa(S_j)$, $S_i = 1$ and $Pr(S_j | Pa(S_j))$ is computed with

the equation (14). $d_j = OR$ if $S_j = 1 \implies \exists S_i \in Pa(S_j),\ S_i = 1$, and $Pr(S_j|Pa(S_j))$ is computed with the equation (15);

$$P_r(S_j|P_a[S_j]) = \begin{cases} 0, \exists S_i \in P_a[S_j] | S_i = 0 \\ P_r(\bigcap_{S_i=1} e_i), otherwise. \end{cases} \quad (14)$$

$$P_r(S_j|P_a[S_j]) = \begin{cases} 0, \forall S_i \in P_a[S_j], S_i = 0 \\ P_r(\bigcup_{S_i=1} e_i), otherwise. \end{cases} \quad (15)$$

– The set of conditional probabilities P defined for each attribute $S_j \in N_{internal} \cup N_{terminal}$ representing the values $Pr(S_j|Pa(S_j))$.

The Bayesian Attack Graph corresponding to the test network is shown in Figure 23 and was constructed from a list of vulnerabilities and the network topology. A static risk analysis requires the system administrator to assess the attacker's capabilities and the difficulty of exploits based on their subjective beliefs. The marginal probability of all attributes S_j is computed from the probabilities $P(S_i)$ of the attributes $S_i \in N_{external}$ and the conditional probabilities of the attributes $S_j \in N_{internal} \cup N_{terminal}$ (cf. figure 22).

A dynamic risk analysis makes it possible to consider the incidents detected on the network during its life cycle by fixing the state of some attributes to true in the BAG. This allows to model the new proofs and to compute the marginal probabilities of all nodes $S_j \in S$. Let $E = \{S'_1, ..., S'_m\} \subset S$ the set of proofs such that $S'_i = 1$ for all $S'_i \in E$, and let the set $S_j \in S - E$ be the set of attributes for which it is necessary to compute the marginal probability.

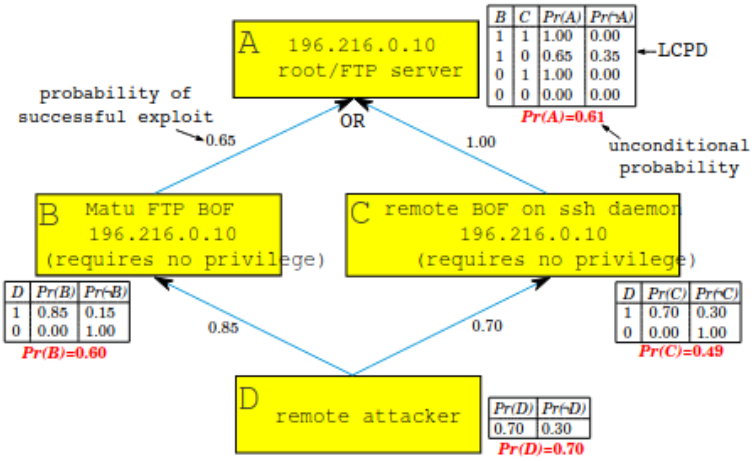


Figure 22: *BAG* simplified to illustrate the computation of probabilities [58]

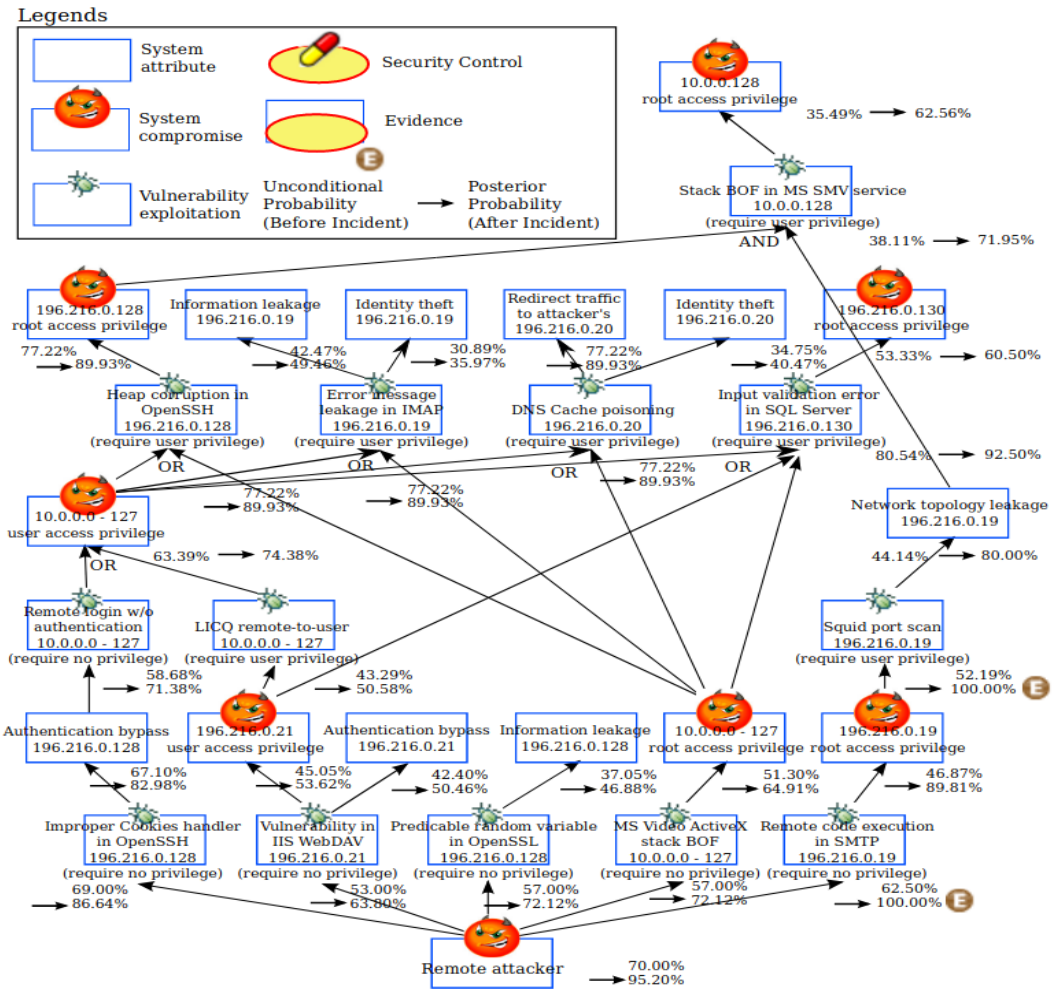


Figure 23: BAG corresponding to the test network with marginal and posterior probabilities [58]

The computation of marginal probabilities is done with the brute force DFS traversal algorithm of complexity $O(2^n)$, and is therefore not adapted for complex systems. However, there are approximate algorithms, such as the Monte Carlo based one [67], which can speed up the computation time, but this has an impact on the accuracy of the results.

Bayesian networks are acyclic graphs. According to the authors this is not a problem because cycles do not increase the probability of success of an attack scenario and can therefore be removed. However, as we have seen in [46], some cycles cannot be removed without impacting the success probability of the attacks (see right-hand graph in Figure 18). Therefore, it seems necessary to use attack trees in the Bayesian network in order to avoid cycles, at the price of increasing the size of the network.

In [64], the authors look for a solution to measure the overall security of information systems. Changes in the nature of vulnerabilities are considered, such as the publication of new technical information, the proposal of a patch or the presence of exploit codes. Dynamic Bayesian Networks (DBNs) are used to model changes in vulnerability properties, measure the evolution of overall system security over time, and make predictions about future system security based on past observations.

An attack graph is defined in this paper as a set $G(E \cup C \cup R_r \cup R_i)$ where E is the set of exploitations, C is the set of security conditions, R_r and R_i are the two sets of relations between conditions and exploitations. The conditional relationships that exist between security conditions and vulnerability exploits in an attack graph G are encoded using a Bayesian network. The TS score measures the difficulty of exploiting a vulnerability by considering its dynamic properties and is computed from CVSS scores using equation (16), with Base Score (BS), Exploitability (E), Remediation Level (RL) and Report Confidence (RC).

$$TS = round\_to\_1\_decimal(BS \times (E \times RL \times RC)) \tag{16}$$

The computation of the probabilities of exploiting the vulnerabilities given that all preconditions are satisfied is computed by normalising the TS score in $[0;1]$ such that $P(e = T | \forall c \in R_r(e), c = T) = TS/10$.

Two types of relationships can be encoded in CPT:

- a disjunctive relationship where only one of the parent nodes needs to be TRUE for the child node to be TRUE;

- a conjunctive relationship where only one of the parent nodes needs to be FALSE for the child node to be FALSE.

Temporal links between random variables of different time slices allow to take into account past events. In this paper, the authors define that if the attacker succeeds in exploiting a vulnerability in one time slice, the probability of exploiting the same vulnerability in future time slices will be equal to 1. The BS and TGS scores are modelled by a node in each time slice. Finally, the probability of a vulnerability being exploited depends on the BS and TGS scores, whether or not the vulnerability was exploited at the previous time slice and the dependency on other vulnerabilities (cf Figure 24).

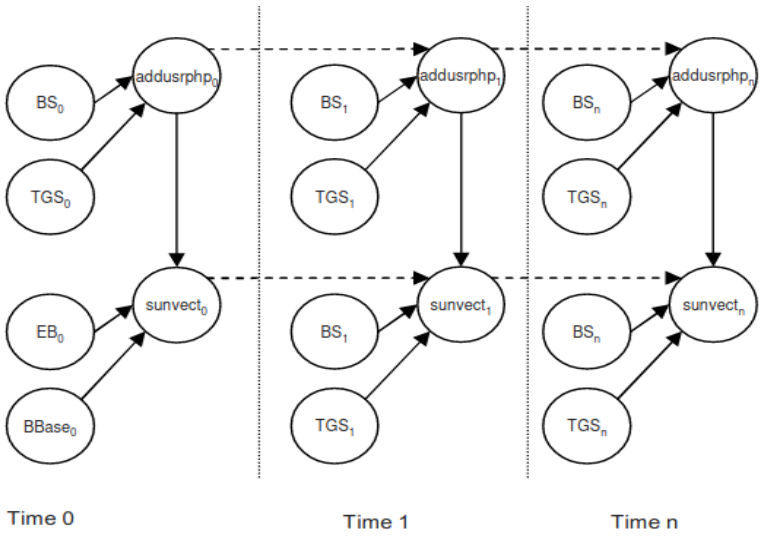


Figure 24: Example of Dynamic Bayesian Network (DBN) for security assessment [64]

The authors did not assess the complexity of the algorithms used, nor performed any benchmark. This is probably not a solution that can be used in large systems without using approximate algorithms. However, the respect of the Markovian properties of the dynamic Bayesian network allows to model it in a compact way.

The uncertainty on the value of conditional probabilities in CPTs can be modeled with noisy gates as in [61]. A noisy AND gate consists in assigning a probability to the TRUE state of a random variable even if one of the dependencies is FALSE. A noisy OR gate consists in assigning a probability to the FALSE state of a random variable even if one of the dependencies is TRUE. It is also possible to aggregate the opinion of several experts to define the values of the conditional probabilities in the CPT as in [62]. The defensive measures implemented by the defenders can also be modeled in the Bayesian network and taken into account when computing the attacker's success probabilities as in [63].

I.3.5 Petri net based attack graphs

Petri nets are graphs where the nodes can be places or transitions. Places can contain tokens and transitions allow, when activated, to move them from place to place. Arcs allow a set of places to be connected to a transition or a set of transitions to a place. A marking represents the distribution of tokens across the places of the Petri net. Petri nets are used to model the evolution of this marking as a function of the activation of the transitions and allow to represent:

- synchronisation;
- parallel processes;
- competition;
- management of resources.

It is possible to check some properties such as:

- reachability: is it possible to reach a given marking given an initial configuration of the Petri net?
- deadlocks: is it possible that the system is in a configuration where no more transitions can be activated?
- boundedness: places are always limited to a quantity M of tokens;
- liveness: no transition will become permanently inactive.

The reachability graph can be generated in order to visualise the evolution of Petri net marking, according to the transitions that are activated. This graph can be infinite, in which case it is possible to generate a coverability graph where the nodes that keep growing in the reachability graph are replaced by a single node ω [68]. There are a multitude of extensions associated with Petri nets such as:

- coloured Petri nets: the tokens can be of several types;
- hierarchical Petri nets: a transition in a Petri net can itself be a Petri net;
- continuous Petri nets: the tokens take real values;

- hybrid Petri nets: the tokens can be either natural or real numbers;
- Petri nets with queues: it is possible to add queues before the places where the tokens will have to stay for a determined time before reaching the place;
- temporal Petri nets: transitions take a fixed time to activate;
- stochastic Petri nets: the transitions take a random time to activate, following an exponential distribution of parameter λ. The average activation time is equal to $1/\lambda$;
- generalized Petri nets: add inhibiting transitions. The transitions can be activated in a deterministic or stochastic time with any random distribution. Immediate transitions can have weights corresponding to activation probabilities.

Petri nets are mainly used to model the attack paths present in a system as in [69]–[73]. In [74], the authors propose a formal method to model the propagation of risk in critical infrastructures. A new class of Petri net called propagation net is defined, which is based on coloured Petri nets and allows to model the relations between the different components of the network structure. In a propagation network, all the components $E = e_1, ..., e_k$ are modelled in the form of places. A place e_i can contain only one token whose value is defined by the function $f(e_i)$ such that $f : E \to V$. The set V represents the feature that is studied in the system. For example it can be a probability and in this case $V \in [0;1]$. The value of the token of a component e_i is computed from the value of the other components on which it depends. This internal dependency is modelled by a function f_i such that $f(e_i) = f_i(f(e'_1), ..., f(e'_k))$, where $e'_1, ..., e'_k$ are the dependencies of e_i, and it is represented by a transition in the propagation network. We note F the set of internal dependencies f_i of the system. The value of some places can also be modified by the environment. We note the set of these places E_b such that $E_b = \{e_1, ..., e_m\}$. An environment is represented by a transition without parents in the propagation network.

In order to propagate the impact of the environment on all the components of the system, the value $f(e_i)$ of a place e_i is updated each time the value $f(e'_i)$ of one of its dependencies e'_i has been modified. To do this, one or more trigger places, noted tr_i^j, are associated with each place representing a component. These places contain a single token of Boolean type and are represented by dotted lines in the propagation network. When a place tr_i^j is true, it indicates that the value $f(e_j)$ of a component e_j must be computed again because the value of its dependency e_i has been modified. The input places of a transition f_i represent the dependencies whose token values are required to compute $f(e_i)$, as well as the trigger place tr_i^j. The places at the output of the transition f_i represent the set of places of type trigger tr_j^k, as well as the place

e_j. When a transition representing the environment is activated, this allows the value of the token of a place representing a component and its associated trigger type place to be modified (see Figure 25). Subsequently, a transition is activated if at least one of the input trigger type places is True. In this case, the following changes are made:

- the value of the token of place e_i is computed with the function f_i;
- the token of the input trigger place is reset to False;
- the token of all output trigger places are set to True if the value $f(e_i)$ has changed.

Propagation stops when all trigger places are False. The propagation network was tested on an example case composed of 5 components $E = \{e_1, ..., e_5\}$, with $V = [0;9]$ and $f : E \rightarrow [0;9]$. Internal dependencies were modeled by experts using the equations (17).

$$\begin{aligned} f(e_1) = f_1(e_1) &= \begin{cases} max(0, f(e_3) - 1) : f(e_1) < 5 \\ f(e_3) : else \end{cases} \\ f(e_2) = f_2(e_2) &= max(f(e_1), f(e_3)) \\ f(e_4) = f_4(e_4) &= (f(e_2) + f(e_3))/2 \\ f(e_5) = f_5(e_5) &= \begin{cases} max(0, f(e_3) - 1) : f(e_5) < 5 \\ f(e_3) : else \end{cases} \end{aligned} \tag{17}$$

An algorithm is used to automatically build the propagation network. The propagation network visible in Figure 27 is used as an example. Two tests were performed:

- the transition *env*1 is activated in order to change the value of the token of place e_1 from 3 to 4. The initial marking of the network is equal to $(3,3,0,1,0,F,F,F,F,F,F)$. The reachability graph is generated and can be seen in Figure 26;
- the transition *env*3 is activated in order to change the value of the token of the place e_3 from 0 to 2. The initial marking of the network is equal to $(3,3,0,1,0,F,F,F,F,F,F)$. The reachability graph is generated and can be seen in Figure 28.

The reachability graph allows to visualise the propagation of the risk through the system until a stable state is reached if it exists. Indeed, in some cases there is no stable state, for example if a cycle is present in the Petri net such that $f(e_{i+1})$ is an argument of f_i, $f(e_{i+2})$ is

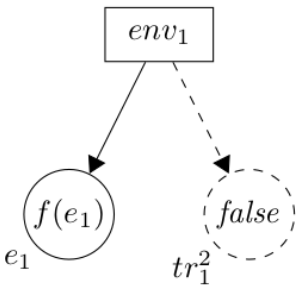


Figure 25: Transition representing the environment [74]

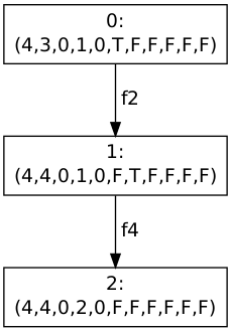


Figure 26: Reachability graph after activation of the transition *env*1 [74]

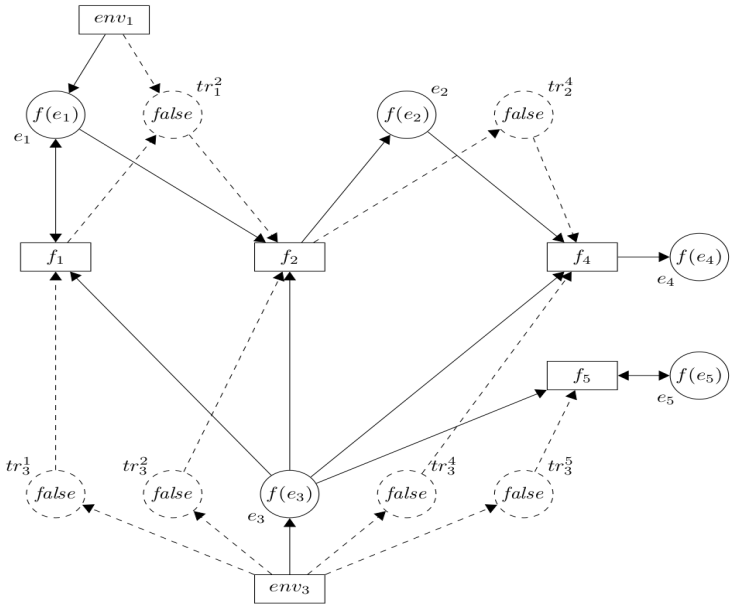


Figure 27: Example of a propagation network [74]

an argument of f_{i+1}, and $f(e_k)$ is an argument of f_i. It also allows model checking to be carried out in order to verify the veracity of some properties of the system.

But unlike what the authors say, model checking algorithms are not adapted to this type of system. Indeed nuXmv is able to analyse systems with more than 10^{20} states, but this does not correspond to a complex system. For example, a simple system composed of 20 elements with 10 possible states gives 10^{20} possible states. It is therefore not able to analyse an information system composed of several thousands machines.

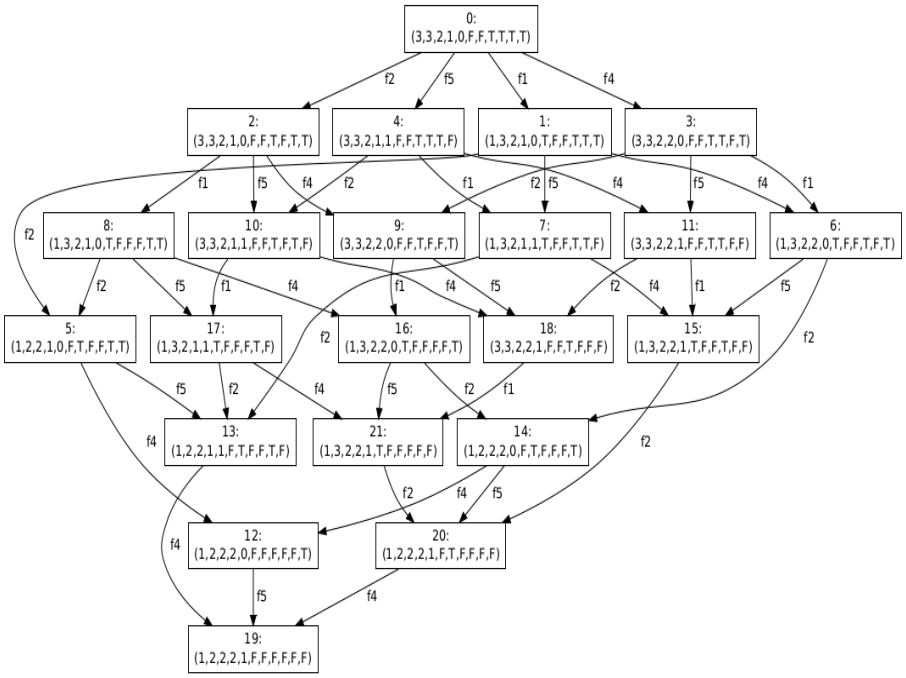
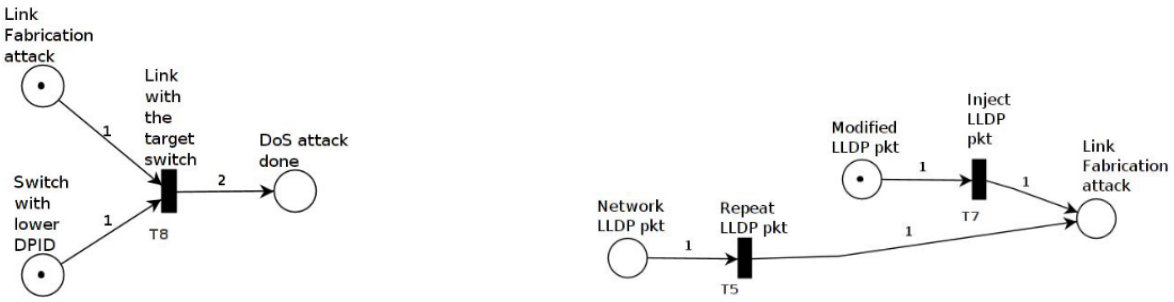


Figure 28: Reachability graph after activation of the transition *env*3 [74]

It is also possible to perform simulations in a Petri net to calculate the chances of success of an attacker. In paper [75], the authors aim to understand and quantify the cyber risk affecting Software-Defined Networking (SDN) using Generalized Stochastic Petri Nets (GSPN). SDN is a network paradigm where the configuration of the network is organised by a controller, which allows the physical separation of network control from the network infrastructure. The weights on the transitions allow probabilistic simulations to be performed. The GSPN places represent the security state and conditions of the attacker, while the transitions represent the actions of the attacker. The input places of a transition represent the preconditions, and the output places represent the results of the action. The tokens in the places represent the different attackers in the system and their progress. A methodology for the construction of the GSPN is proposed:

- define the attacker's main objective and sub-objectives;
- list all possible security states of the SDN components;
- identify all possible actions of the attacker that could change the security state of the components;
- build the model from the previous information that allows the attacker's main objective to be achieved.



(a) Logical AND relationship

(b) Logical OR relationship

Figure 29: Logical relationships in a Petri net [75]

A Boolean AND relationship can be modelled in a GSPN (see Figure 29a) and allows to represent the case where at least one token is required in each place to activate the transition. A Boolean OR relation can be modelled (see Figure 29b) and allows to represent the case where a precondition with at least one token is required to activate the transition.

Place	**Description**
P_0	DoS attack start
P_1	Attack route 1
P_2	Attack route 2
P_3	Genuine LLDP packet
P_4	Network LLDP packet
P_5	DPID of two switches
P_6	Modified LLDP packet
P_7	Link Fabrication attack
P_8	Switch with lower DPID
P_9	DoS attack done
Transition	**Description**
T_0	Launch DoS attack
T_1	Get genuine LLDP packet from external source
T_2	Get LLDP packet from one target switch within the network
T_3	Listen to LLDP packets
T_4	Modify LLDP packet
T_5	Repeat LLDP packet
T_6	Find the lower DPID
T_7	Inject LLDP packet
T_8	Link with the target switch
T_9	Restart DoS attack

Table 7: List of places and transitions in the Petri net

The authors tested their model on attacks targeting the topology management service used by the SDN controller. The network discovery protocol works like this:

1. the controller sends a Link Layer Discovery Protocol (LLDP) packet to a switch S_1. This packet contains the port ID and the Datapath ID (DPID) of the switch. The DPID is used to identify the switch to the controller. The LLDP packet is encapsulated in a Packet-Out message.

2. the switch then broadcasts this packet on all its ports.

3. when a switch S_2 receives the message on a port that is not connected to the controller, the LLDP packet is encapsulated in a Packet-In message that is then sent to the controller. The LLDP packet contains the port ID and DPID of the switch S_2.

4. when the controller receives the message, it can detect the link between switches S_1 and S_2 and update the network topology accordingly.

The GSPN visible in Figure 30 allows to represent the attack paths allowing to realize a denial of service attack on the topology management service. The different places and transitions are listed in Table 7. The places P_0, P_1 and P_2 represent the initial states. The transition T_9 allows to restart the attack in order to perform several simulations. The probabilities on the transitions are computed according to three parameters:

- the cost of the attack c_A for the attacker;
- the technical difficulty d_A;
- the discovering difficulty s_A.

Attack Cost (10^3) c_A	**Grade**	**Technical difficulty** d_A	**Grade**	**Discovering Difficulty** s_A	**Grade**
> 10	5	Very difficult	5	Very difficult	1
$6-10$	4	Difficult	4	Difficult	2
$3-6$	3	Medium	3	Medium	3
$0.5-3$	2	Simple	2	Simple	4
< 0.5	1	Very simple	1	Very simple	5

Table 8: List of possible values for the different parameters

The possible values of these parameters are shown in Table 8. The probability P_A of a transition is computed from the following equation:

$$P_A = w_1 \times u_1(c_A) + w_2 \times u_2(d_A) + w_3 \times u_3(s_A) \tag{18}$$

Transition	Attribute			Occurrence probability
	Attack cost c_A	Technical difficulty d_A	Probability to be discovered s_A	
T_1	2	1	5	0.113
T_2	3	2	3	0.078
T_3	1	2	4	0.117
T_4	3	2	3	0.078
T_5	2	2	2	0.10
T_6	1	1	5	0.147
T_7	3	4	2	0.072
T_8	4	4	2	0.067

Table 9: Values of the different parameters of the transitions

with $w_1 + w_2 + w_3 = 1$ the weights of the different parameters, and $u(x) \in [0;1]$ a utility function of the parameters associated with the transitions. In this model, $w_1 = w_2 = w_3 = 1/3$ and $u_1(c_A) = u_2(d_A) = u_3(s_A) = 0.2/x$. The probabilities of all the transitions in this model are shown in Table 9.

The open source tool PIPE [76] (Platform-Independent Petri net Editor) is used to model and analyse the GSPN corresponding to the use case shown in Figure 30. A weight of 1 is assigned to the transitions T_0 and T_9. The reachability graph that is generated makes it possible to explain the behaviour of the system, to verify the absence of deadlock, and to guarantee it is bounded.

50 simulations are performed with each time a different number of activation of the initial transition $T0$. Figure 31 shows the distribution of the tokens in the Petri net and allows to approximate the success probabilities of the attacker. For example, the attacker has a 17% chance of succeeding in the $P7$ attack and a 29% chance of succeeding in the $P8$ attack.

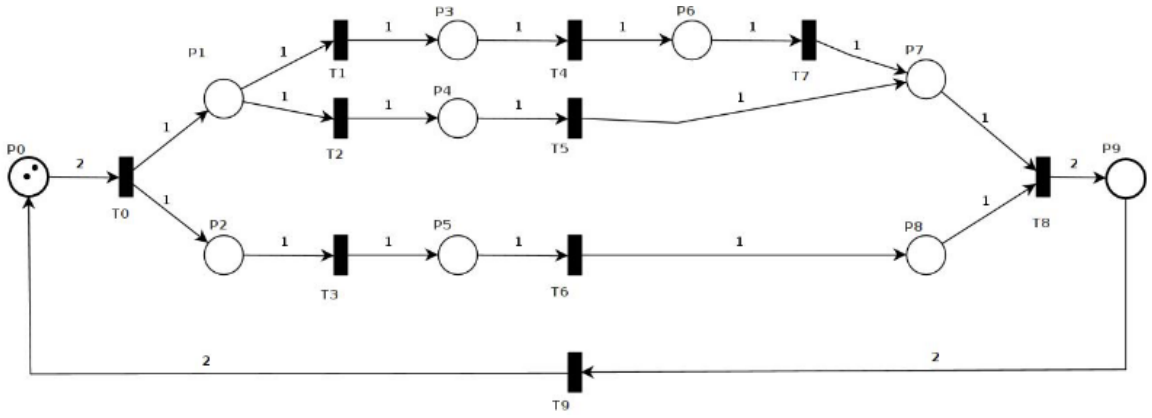


Figure 30: Petri net used in the use case [75]

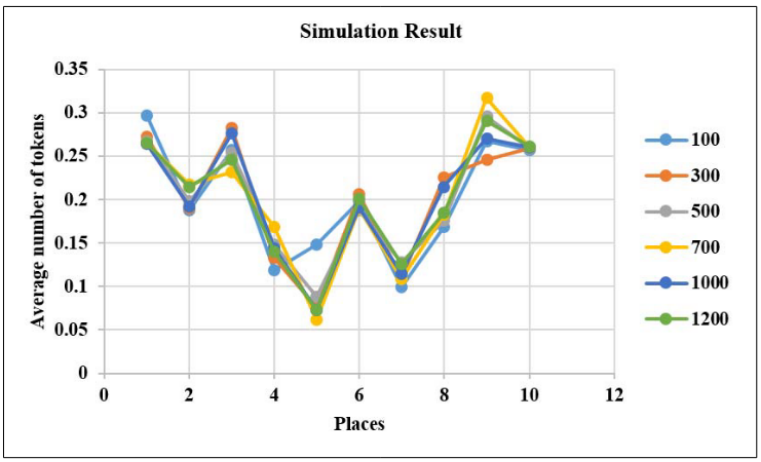


Figure 31: Distribution of tokens in the Petri net [75]

The solution proposed in this paper allows to compute the chances of successful compromise by an attacker. The probabilities are approximated by performing several simulations in the Petri net, which allows to use this method in a complex system. However, the method proposed by the authors to build the Petri net relies on too much human work and it would be interesting to verify if an attack graph generated automatically with a tool such as MulVAL cannot be transformed into a generalized Petri net.

The modeling of attack paths can be simplified by using hierarchical Petri nets as in [73], [77]. In this network, a node of the first layer represents the exploitation of a vulnerability and this node can be associated with a second Petri net of a lower layer to detail the steps necessary for the exploitation.

I.3.6 Conclusion

The modeling of attack paths allows to represent in a very precise way all the actions that the attacker can perform to compromise the system. The attack graph can be generated automatically with algorithms such as MulVAL or NetSPA from a detailed description of the system and the attacker's capabilities. These algorithms are suitable for large systems with respective worst-case time complexity $O(n^2 \times log(n))$ and $O(max(V,T) \times R \times C)$, where n is the number of components in the system, V the number of vulnerabilities, T the number of ports, R the number of reachability groups and C the number of credentials. It is also possible to take into account the difficulty of exploiting the vulnerabilities and to calculate the success probabilities of the different actions of the attacker. However, the accuracy provided by this model requires an accurate and a comprehensive modeling of the system, which can be difficult

to achieve. Especially when some information cannot be known with sufficient confidence, such as information about the real capabilities of the attacker. The complexity of the calculations of the attacker's success probabilities can also be in some cases too complex to be used in large systems, such as inference calculations in Bayesian networks [58] or model checking in Petri net [74]. Moreover, the modeling of attack paths is often based on a static representation of the system. The only exception are the solutions based on dynamic Bayesian networks as in [64], which allows to represent the evolution of the CVSS score of a vulnerability. To summarize, the modeling of attack paths allows a very accurate analysis of the risk of compromise of the system by an attacker, but a lot of information is required to model all the properties of the system and the capabilities of the attacker. Especially since some of this information can be difficult to obtain and its collection cannot always be done automatically.

I.4 Epidemiological models

I.4.1 Definition

An epidemiological model is used to study the evolution of an infection in a population over time. The population is divided into several groups whose number and meaning may change depending on the models used. For example, the SIR (Susceptible, Infected, Recovered) model represents three population groups: susceptible, infectious and recovered. A person who is susceptible to infection can be moved into the infectious group when he or she contracts the virus. An infectious person can pass into the recovered group when the infection is over and he or she becomes immune to the virus. The SIS (Susceptible, Infected, Susceptible) model is identical to the SIR model except that there is no immunity: an infectious person, once recovered, returns to the group of people susceptible to be infected. The SEIS (Susceptible, Exposed, Infected, Susceptible) model is an extension of the SIS model where a new group is created corresponding to people who have been infected but cannot yet transmit the virus. A susceptible person can move into the exposed group when he or she is infected with the virus. An exposed person enters the infectious group when he or she can transmit the virus. The evolution of populations within these different groups of individuals can be represented in a deterministic way with differential equations or in a stochastic way by modeling the random behavior of the virus. Epidemiological models can be used to model the propagation of an attack through a computer system.

I.4.2 Deterministic assessment

In [78], the authors study the propagation of jamming attacks on IoT wireless networks. A jamming attack consists in saturating the communication channel of a device. This attack can be performed at the physical or Medium Access Control (MAC) layer of the Open Systems Interconnection (OSI) model, and does not require a lot of computing power. The symptoms of this attack are an increase in the number of packets transmitted between nodes, packet loss and an extra consumption of resources. An infected node will produce the same effects on its neighboring nodes.

In this paper, the authors use a *SIR* epidemiological model to study the spread of the attack. In a *SIR* model with N people, three quantities evolve over time: the number S of people susceptible to be infected, the number I of infected and the number R of recovered. The number of new infections depends on the infection rate β, which is the average number of attacks per unit of time. The period of recovery γ depends on the duration of the infection t_d. A *SIR* model is defined such as:

$$N(t) = S(t) + I(t) + R(t) \tag{19}$$

$$\begin{aligned} \frac{dS(t)}{dt} &= -\beta I \frac{S}{N} \\ \frac{dI(t)}{dt} &= \beta I \frac{S}{N} - \gamma I \\ \frac{dR(t)}{dt} &= -\gamma I \end{aligned} \tag{20}$$

$$\gamma = \frac{1}{t_d} \tag{21}$$

The steady state of the system such as $dS(t)/dt = 0$, $dI(t)/dt = 0$, $dR(t)/dt = 0$ allows to obtain the reproduction rate R_0. This metric indicates the average number of new infections caused by an individual during its own infection period. If $R_0 \leq 1$, then

$$\lim_{t \to \infty} I(t) = 0 \tag{22}$$

If $R_0 > 1$, then

$$\lim_{t\to\infty}[S(t),I(t),R(t)] = (\frac{N}{R_0}, \frac{\beta N}{\beta+\gamma}(1-\frac{1}{R_0}), \frac{\gamma N}{\beta+\gamma}(1-\frac{1}{R_0})) \tag{23}$$

The maximum number of infected I_{max} is computed as follows

$$I_{max} = S(0) + I(0) - \frac{\gamma}{\beta} ln(S(0)) - \frac{\gamma}{\beta} + \frac{\gamma}{\beta} ln(\frac{\gamma}{\beta}) \tag{24}$$

The authors focus on two types of attacks: (1) reactive mode attacks where packets are sent out by the infected node as soon as a transmission is detected. (2) Random attacks when packets are sent for a period of time t_j before going into sleep for a period of time t_s. The wireless IoT network is modeled by a graph where the nodes represent devices of same nature and the links represent radio connections between them. A link exists between two nodes if their distance is less than or equal to the range of their radio transmission r_0. The nodes are uniformly distributed on a two-dimensional area A. The average density ρ of a network of N nodes is therefore equal to $\rho = N/A$. The degree of a node corresponds to its number of connections. The probability that a node is connected to k neighbors is

$$p(k) = \binom{N-1}{k} p^k (1-p)^{N-1-k} \tag{25}$$

with

$$p = \frac{\pi r_0^2}{A} \tag{26}$$

The objective is to consider the density of the system while modeling the epidemic. Two metrics are defined:

- the severity of the attack λ such that $\lambda = \rho \times \beta$;
- the persistence of the attack t_d such that $t_d = 1/\gamma$.

The differential equations and the computation of I_{max} are redefined to take into account the density of the system:

$$\begin{aligned} \frac{dS(t)}{dt} &= -\lambda I \frac{S}{N} \\ \frac{dI(t)}{dt} &= \lambda I \frac{S}{N} - \gamma I \\ \frac{dR(t)}{dt} &= -\gamma I \end{aligned} \tag{27}$$

$$I_{max} = S(0) + I(0) - \frac{\gamma}{\lambda} ln(S(0)) - \frac{\gamma}{\lambda} + \frac{\gamma}{\lambda} ln(\frac{\gamma}{\lambda}) \tag{28}$$

The epidemiological model is validated using experimental data from study [79], [80], where three different routing protocols are compared: Multi-Parent Hierarchical Protocol (MPHP), Ad-hoc On-demand Distance Vector (AODV) and Dynamic Source Routing (DSR). A grid of dimension 300×300 is used where 49 static nodes are placed randomly following a uniform distribution. Each node has a radio range of 50m. A coordinator node is placed in the upper left corner and collects all the information generated by the other nodes of the network. Different experimental conditions were tested:

- the position of the attacker's node (top left, middle and bottom right);
- the type of attack (random attack with a rate of 50 packets/s, random attack with a rate of 80 packets/s, reactive attack);
- the type of routing protocol (AODV, DSR, MPH).

The performance of the network is evaluated over a period of 100 seconds with 270 samples by measuring the number of nodes that the coordinator is able to reach. A node is able to communicate with the coordinator if it has not been infected or if the infection has ended. A tool has been developed to compare experimental data and differential equations. This tool also allows to compute attack severity R_0 and attack persistence $1/\gamma$ values from the experimental data.

Theoretical and experimental results are compared on the basis of attack severity, attack persistence and peak attack value. The authors find that the theoretical values are close to the experimental values. However, there is a difference in the case of the MPH protocol when the primary infection node is near or far from the coordination node, and in the random attack with a rate of 50 packets/s.

The authors note several things that need to be improved: (1) new experimental data must be generated. These data are required to validate the theoretical model and there are currently not enough of them. An environment to simulate jamming attacks could be developed in the future. (2) Other node distributions should be considered. In this study, only a uniform distribution was considered. (3) The stochastic behavior of the system must be modeled.

This solution is only applicable for one type of attack. It is undoubtedly possible to adapt it to other types of attacks with different impacts than the unavailability of the component. Several points of concern can be made in this study: (1) the authors did not explain how the parameters β and t_d were obtained when solving the differential equations. (2) The authors chose to compare the experimental and theoretical results using indicators such as attack severity and attack persistence. It would have been more relevant to compare them by computing a correlation coefficient.

I.4.3 Stochastic assessment

Epidemiological models can also take into account the stochastic behavior of the system when calculating the propagation of an attack. In [81], the authors use epidemiological modeling to assess the spread of a generic problem through a system by studying the cascading effects that can occur. For example, it can be the propagation of an impact through the chains of dependencies or the propagation of a virus in the system.

A stochastic model is required to model random phenomena such as the connection of user devices to workstations (Bring Your Own Device - BYOD). Stochastic models are built from a large amount of observational data, which may be difficult to obtain or unavailable, especially in a cyber security context. It is therefore necessary that a stochastic model works with little data.

An example case is defined, which corresponds to the case of the propagation of a malware in a network infrastructure. This attack is similar to the Stuxnet [10] attack, where a worm penetrated the system through a USB device and spread through the network.

A network infrastructure is modelled with a graph $G = (V, E)$ with V the set of nodes and E the interconnections such that $E \subseteq V \times V$. Different classes of connections are defined according to their properties to transmit a problem. E is thus partitioned in m subsets such that to each link of class k is associated:

- a probability p_k of transmitting a problem;

- an infection rate λ_k corresponding to the average number of events per unit of time. For example, this could be the average number of emails received each hour. This data can be obtained by analysing network traffic, counting emails, or by asking users how often they use a given communication method.

The authors model the number of infection attempts by a Poisson distribution of parameter λ_k, and the waiting time between two events by an exponential distribution of parameter $1/\lambda_k$. An algorithm allows to simulate the propagation of a problem (cf. Algorithm 1). All the nodes of the graph are initially coloured in green. A node $v_0 \in V$ is coloured in red and represents the entry point of the infection. At each step of the simulation, all infected nodes are traversed and propagate the problem through a link with probability p_k. A simulation time limit T is defined, and at each propagation attempt, the time counter t is incremented by e^{1/λ_k}. The result of the algorithm is a partially coloured graph where the nodes in red correspond to the infected nodes. The infection rate can be calculated and is equal to $\frac{N}{|V|}$ where N is the number of infected nodes. If M nodes have been recovered, then the infection rate is $\frac{N-M}{|V|}$. The amount M depends on the recovery processes of the organisation.

Algorithm 1 Algorithm for simulating the propagation of a problem

Input: G = (V,E)
Output: Color nodes of the graph
$t \leftarrow 0$
while $t < T$ **do**
  for each red node in V, set $N(v) \leftarrow u \in V : (v,u) \in E$ **do**
    for each neighboring node $u \in N(v)$ **do**
      let k be the class in which the edge $v \rightarrow u$ falls into
      with likelihood p_k, color u in red
      draw an exponentially random $\Delta t \simeq e^{1/\lambda}$
      $t \leftarrow t + \Delta t$
    end for
  end for
end while

The value p_k can be obtained by aggregating several expert evaluations in order to take uncertainty into account. Let $p_{k,1}, ..., p_{k,n_k}$ be the set of n_k estimates from the different experts. The value p_k can be defined from a random variable X_k which follows the distribution of the different expert opinions (see Figure 32). The probability p_k is obtained from equation (29), which represents the proportion of experts who consider the probability p_k higher than 0.5.

$$p_k := Pr(X_k > 0.5) \tag{29}$$

A pandemic criterion is defined from the percolation theory [82] and allows to affirm or not that the propagation of the problem will remain local to a part of the system or will spread to all the parts of the system. If the network is described through an Erdös-Rényi [83] model with n nodes, m classes of links, q_i the probability that a link of class i exists, and p_i the probability

that a link of class i propagates the problem, then the propagation remains local to a part of the system if equation (30) is satisfied.

$$1 - n \times p_1 \times q_1 - \ldots - n \times p_m \times q_m \tag{30}$$

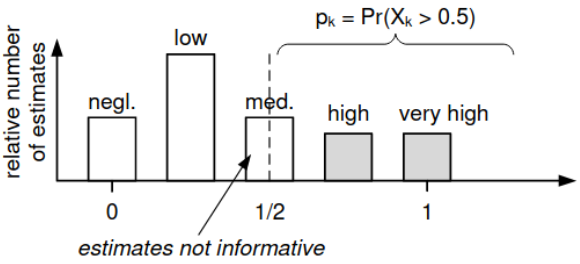


Figure 32: Distribution of the different experts' opinions for the value p_k [81]

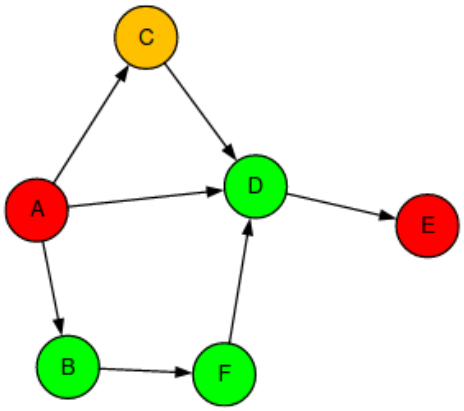


Figure 33: Heatmap of the system [81]

The result of the simulations can be visualised in the form of a heatmap (see Figure 33), where each node is coloured between purple and green according to the probability of infection. This probability is computed by dividing the number of times a node has been infected by the number of simulations performed. The authors propose to perform simulations for different security criteria such as unavailability or integrity, in order to obtain a different heatmap in each case. However, it does not seem possible to model dependencies between several security criteria, such as the loss of data integrity leading to the unavailability of a component.

I.4.4 Conclusion

Epidemiological models allow to represent the evolution of a system when an attack is launched by a malicious actor by measuring the evolution of the number of components impacted by the attack. In the case of deterministic modeling, differential equations are used to represent the evolution of the number of infected components over time. In the case of stochastic modeling, probabilities are associated with the infection rates to represent differences in infection success based on the transmission context of the virus. Simulations are then performed from the differential equations to approximate the evolution of the impact of the attack on the different components of the system. These simulations make it possible to an-

alyze large systems in a reasonable amount of time. In the case of stochastic modeling, it is possible to display a heatmap of the system to visualize the components most impacted by the attack. However, epidemiological models are mainly used to represent the propagation of a single type of impact through the system. More attention is required to adapt this model to the representation of an attack mixing loss of integrity, confidentiality and availability.

I.5 Complex network theory

I.5.1 Definition

The complex network theory allows to study the structure of complex systems. Different metrics are defined to analyze the properties of nodes and edges of the network. A system can be represented by a graph where nodes represent atomic elements with some properties, and edges represent relationships between them. An edge can be oriented and a weight can be assigned to it. A graph can be extended into a multigraph where several edges are possible between two nodes, or into a multidimensional graph where several levels of the graph are modelled, such as bipartite graphs. The size of a graph corresponds to its number of nodes.

A walk is a sequence of edges between two nodes where repetitions are allowed, unlike a path where repetitions are not allowed. The shortest path between two nodes is called a geodesic. The diameter of a network is its longest geodesic.

There are different network topologies that will have an impact on its properties such as trees, rings, starts, complete graphs, lines, meshes or buses. The adjacency matrix is a matrix of dimension 2 where $a_{ij} = 1$ if there is an arc from node i to node j, and 0 otherwise. In an undirected graph, $a_{ij} = a_{ji}$. If there is a weight w_{ij} on an arc, then $a_{ij} = w_{ij}$.

The degree of a node measures the number of connections it has with other nodes, and is computed with the following equation:

$$ND_i = \sum_{j=1, j\neq i}^{N} a_{ij} \tag{31}$$

This definition can be extended for directed graphs by computing the In-degree with equation (32) and measuring the number of arcs going to the node.

$$ND_i^{in} = \sum_{j=1, j\neq i}^{N} a_{ji} \tag{32}$$

The Out-degree is computed with equation (33) and measures the number of arcs outgoing from the node.

$$ND_i^{out} = \sum_{j=1, j\neq i}^{N} a_{ij} \tag{33}$$

A node with a high degree can become a hub of the network. The degree distribution of a graph represents the frequency distribution of the different degrees of the nodes.

The density of a network is calculated with equation (34) and measures the ratio between the number of connections in the network and the total number of possible connections, with n the number of nodes in the graph.

$$D = \frac{\text{Actual Connections}}{\text{Potential Connections (PC)}}, \text{ with } PC = \frac{n \times (n-1)}{2} \tag{34}$$

The clustering coefficient measures the level of aggregation of nodes in a network. There is a global coefficient that measures the global aggregation of the nodes in the network and is calculated with equation (35), where ND_i the degree of node i and V the set of nodes of the graph.

$$C = \frac{3 \times |triangles|}{\sum_{i \in V} \binom{ND_i}{2}} \tag{35}$$

There is also a local coefficient that measures the level of aggregation around a node v_i. This coefficient is calculated with equation (36).

$$C_i = \frac{3 \times |\text{triangles with vertex i}|}{\binom{ND_i}{2}} \tag{36}$$

The mean of the local clustering coefficients of the graph nodes can be computed with the following equation:

$$\frac{1}{|V|} \times \sum_i c_i \tag{37}$$

A network can have subsets of particular nodes (see Figure 34) such that:

- Component: a set of nodes such that for all nodes x and y, there exists a path $P(x,y)$. And this subset of nodes is not connected to the rest of the network.
- Cluster: a subset of nodes that are strongly connected and share common properties.
- Clique: a subset of nodes in a graph where there is one edge between each pair of vertices.

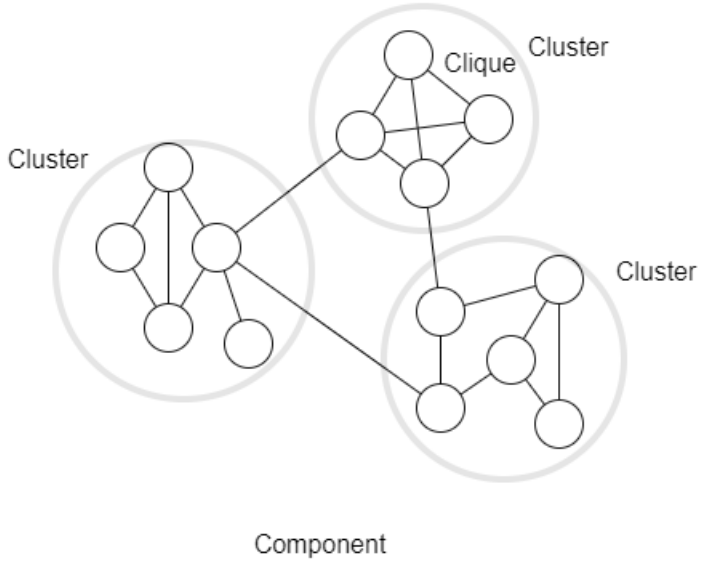


Figure 34: [84] Representation of a clique, clusters, and a component

The importance of a node in the network can be measured by computing different metrics such as:

- degree connectivity: measures the number of connections with other nodes in the network.
- closeness centrality: measures the ability of the node to affect all others:

$$C(x) = \sum_y \frac{1}{d(x,y)} \tag{38}$$

- betweenness centrality: measures the number of shortest paths that pass through this

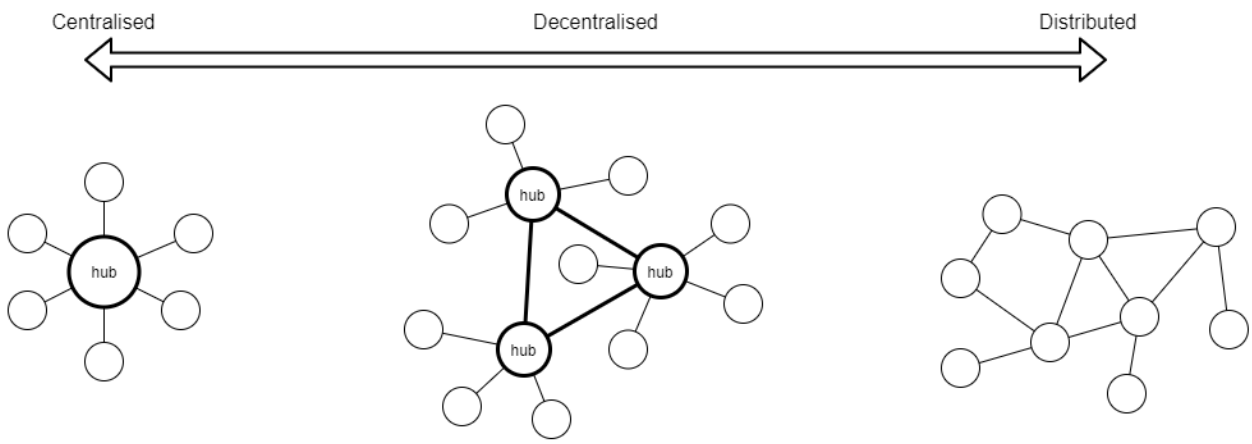


Figure 35: The range of distribution degrees

node:

$$BC(x) = \sum_{x \neq y \neq z}^{N} \frac{\omega_{yz}(x)}{\omega_{yz}} \tag{39}$$

, with $\omega_{yz}(x)$ the number of shortest paths between y and z passing through x, and ω_{yz} the number of shortest paths passing between y and z.

- flow betweeness centrality: same as Betweeness centrality but for all paths.
- prestige centrality: the importance of the node depends on the importance of its neighbours, so we obtain a recursive definition of centrality. For example, the eigenvector centrality allows to propagate the information of the degree centrality to the whole network. To do this, we define an initial vector $X = [1, 1, ..., 1]$, then we multiply X by the adjacency matrix A, until we reach a stable state where the values of the matrix X all increase at the same rate. We thus obtain $AX = \lambda X$. This equation can be solved by finding the eigenvalues and eigenvectors of the system.

There are mainly three types of networks (see Figure 35):

- distributed: the distribution of degrees in the network follows a normal distribution. The network is super resilient.
- decentralised: a set of clusters linked together at the level of hubs. It respects the small world property: most nodes are not neighbours but the distance between them is small.
- centralised: there is an exponential relationship between the degree of connectivity of a node and its frequency of occurrence. This is called a scale-free network and it is very sensitive against targeted attacks.

The next section will show how the previously defined metrics can be used to assess the impact of a vulnerability on the overall security of the system. For this purpose, two papers are presented: the first one is interested in evaluating the impact of a cyber attack on the cyberspace by modeling the cascading effects following the deletion of a node in the network. The second article focuses on the survivability of a powergrid when some relays become unavailable, either because of an attack or because of the environment.

I.5.2 Cyberspace survivability

In [85], the authors attempt to measure the impact of network topology on the survival of cyberspace using complex network theory. A cyberspace is defined as a large network such as Internet. Different attack strategies based on metrics from complex network theory are defined:

- random attack on a node.
- random attack on a link.
- deliberate attack based on the static degree of a node.
- deliberate attack based on the dynamic degree of a node.
- deliberate attack based on the betweenness centrality of a node.
- deliberate attack based on the betweenness centrality of a link.

An attack consists of removing a set of nodes or links from the network, modeling their unavailability as a result of the attack. Attacks can be deliberate or random. Random attacks can represent natural phenomena or human errors. The objective is to identify the attack strategy that has the greatest impact on the system, and to use the corresponding metric to identify the critical points of the network. The exploitation of a vulnerability at a critical point in the network will therefore have a strong impact on the survivability of cyberspace.

Different metrics are defined to measure the impact of an attack:

- information transmission efficiency index E in cyberspace. This metric measures the difference in the sum of the reciprocal of the shortest transmission paths in cyberspace with

and without cyber attack, and is computed with equation (40):

$$E = \frac{\sum_{i \in G, j \in D} \frac{1}{\omega_{ij}}}{\sum_{i \in G^0, j \in D^0} \frac{1}{\omega_{ij}^0}} \tag{40}$$

with G the set of service nodes, D the set of terminal nodes, ω_{ij} the shortest transmission path between the service nodes i and the terminal nodes j, and the exponent ^{0} when it is the context without attacks.

- index of maximum connected regions in cyberspace Q. This metric measures the number of nodes present in the largest remaining connected part of the network and is computed with equation (41):

$$Q = max(\sum Set_m), m = 1, 2, ..., k \tag{41}$$

with Set_m the different sets of connected nodes and k the number of sets.

The evolution of these indices is measured according to the attacks carried out on the network. Tests were performed on a system with 2145 nodes and 3260 links. Figure 36 shows the evolution of the information transmission efficiency index E as a function of the number and different types of attacks performed on the network nodes, while Figure 37 shows the evolution of the index of the largest connected subnetwork Q.

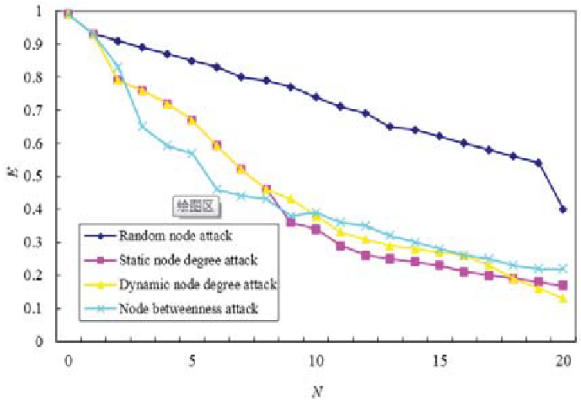


Figure 36: Evolution of the information transmission efficiency index as a function of the number of attacks performed on network nodes [85]

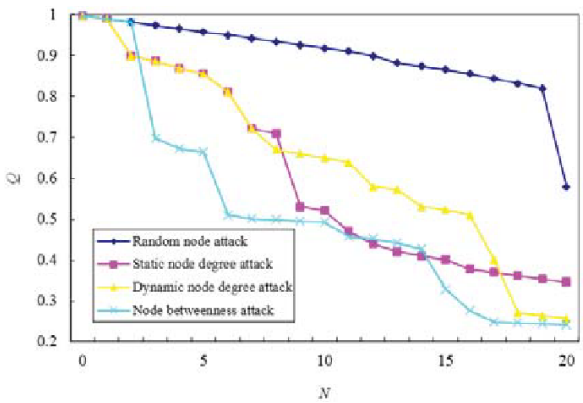


Figure 37: Evolution of the index of the largest subnetwork as a function of the number of attacks performed on network nodes [85]

The analysis of these data allows to notice that the attack strategy that consists in dynamically targeting nodes with a high degree of connectivity is the most effective in damaging the efficiency of the information transmission in the network, while the strategy targeting nodes with a high betweenness centrality is the most effective in fragmenting the network.

Figure 38 shows the evolution of the information transmission efficiency index E as a function of the number and different types of attacks performed on the network links, while Figure 39 shows the evolution of the index of the largest connected subnetwork Q.

The analysis of these data allows to notice that the attack strategy which consists in targeting the links with a strong betweenness centrality is the most effective to damage the efficiency of the transmission of the information in the network as well as to fragment it.

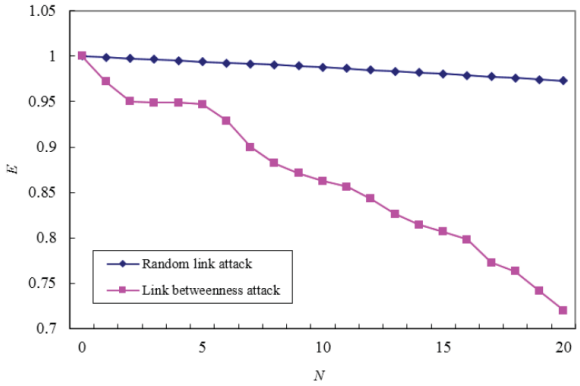


Figure 38: Evolution of the information transmission efficiency index as a function of the number of attacks performed on network links [85]

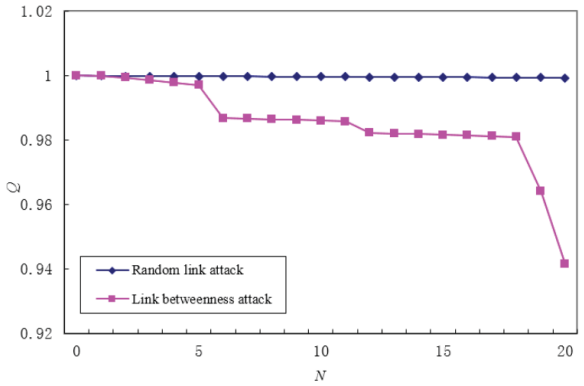


Figure 39: Evolution of the index of the largest subnetwork as a function of the number of attacks performed on network links [85]

Exploiting a vulnerability on a highly connected host will therefore have a strong impact on the proper functioning of the system, and defensive measures should be applied as a priority on these nodes.

I.5.3 Powergrid survivability

In [86], the authors aim to identify the most critical assets of the Indian electricity network in order to protect them as a priority and thus increase the robustness and reliability of the network. Indeed, attacks on these assets can make them unavailable and cause a cascading effect that can lead to a blackout.

Attacks can be deliberate or random. Random attacks can represent natural phenomena such as thunderstorms, extreme winds, or floods. The power system is represented by a graph (see Figure 40) where the nodes correspond to the substations in the system, and the edges represent the transmission lines. Weights are associated with the edges to model the transmission line reactance. Reactance is a form of electrical resistance that is caused by changes in current in an Alternating Current (AC) circuit. More power can be transmitted through a line with a low

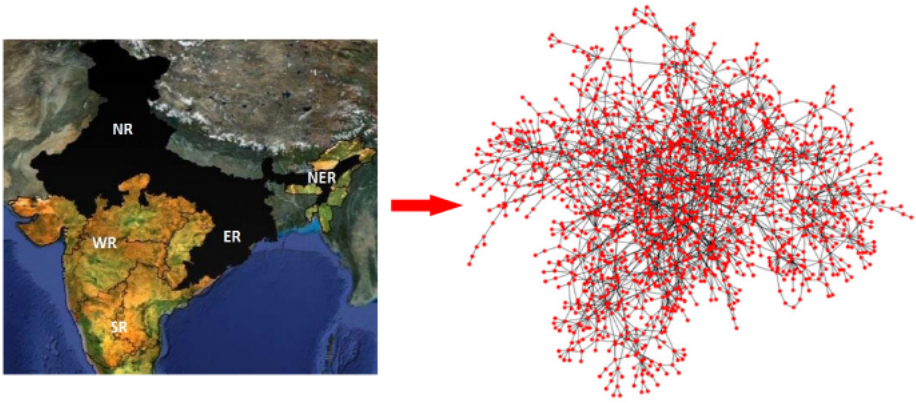


Figure 40: Graphical representation of the Indian electricity network [86]

reactance.

Several metrics from complex network theory are adapted to weighted graphs. The degree of a node ND_i is computed with equation (42), with N the number of nodes in the graph and w_{ij} the weight associated with a edge between node i and j. The calculation of the shortest path between two nodes is also redefined to take into account the weights associated with the links and is equal to the smallest sum of weights of lines connected between that two nodes.

$$ND_i = \sum_{j=1, j\neq i}^{N} w_{ij} \tag{42}$$

Different attack strategies are defined:

- random attacks (RA);
- static attacks on links with the highest betweenness centrality (SL);
- dynamic attacks on links with the highest betweenness centrality (DL);
- attacks on nodes with high betweenness centrality of low reactance paths (HBNL);
- static attacks on the nodes with the highest degree centrality (HDNS);
- dynamic attacks on nodes with the highest degree centrality (HDND);
- static attacks on nodes with the highest betweenness centrality (HBNS);
- dynamic attacks on nodes with the highest betweenness centrality (HBND).

Different metrics are defined to measure the performance of the network:

- the ratio S between the number of nodes in the largest subgraph before deletion (N_a) and after deletion (N_i):

$$S = \frac{N_a}{N_i} \tag{43}$$

- the efficiency E_f of the network to transmit loads, with ω_{ij} the shortest path between nodes i and j and N the total number of nodes in the network:

$$E_f = \frac{1}{N(N-1)} \sum_{i \neq j} \frac{1}{\omega_{ij}} \tag{44}$$

The model has been tested on two use cases. The first one is the Indian electrical network composed of 1,634 nodes and 2,549 links. The second is an approximation of the American electrical network as it was in 1962 named IEEE 118. Attacks targeting nodes with high betweenness centrality are the most effective in reducing network efficiency.

Complex network theory has identified the most effective attack strategy to degrade the performance of the power system, as well as the critical assets of the system. We will now discuss the advantages and disadvantages of this type of solution for risk analysis in complex systems.

I.5.4 Conclusion

The complex network theory allows to compute metrics from a network representing the system components and their interactions. These metrics are used to measure the impact of an attack on the final network structure. The complexity of the calculations performed are most often adapted to large systems. For example, computing the betweenness centrality of a node requires listing the complete set of shortest paths between all nodes in the network, which can be done with Floyd Warshall algorithm with a worst-case time complexity $O(n^3)$, with n the number of nodes [87]. However, the impact of an attack on the system is represented by the removal of a node or link in the network, which limits the representation of the impact of attacks to the unavailability of a component. It would be interesting to verify if it is possible to adapt this theory to model other types of impacts such loss of confidentiality or loss of integrity.

I.6 State-based modeling

I.6.1 Definition

State-based modeling is a type of modeling that consists in representing the behavior of a system. This modeling makes it possible to represent the dynamic behavior of a system in response to changes in some of its internal variables or in its environment. With this model, it is possible to describe the properties of a system and their evolution. Each individual property of the system can take a finite or infinite set of values. For example, the state of a vehicle can be defined in the finite set $S_1 = \{on, off\}$. But the position of this vehicle on a map can be defined by the infinite set $S_2 = \{(x,y)|x \in \mathbb{R}, y \in \mathbb{R}\}$. The state of a system is a combination of all the values of the different properties that compose it. For example, the state of the previously defined system takes its values in the set $S_1 \times S_2$. For example, the state of the vehicle may be $(on, (11,4))$. The number of states of the system can be infinite when one of its properties takes its values in an infinite set, which is the case in this example. The transitions between the different states of the system can be represented with a graph called Automaton. A Finite Automaton allows to model the behavior of a system whose number of possible states is finite. On the other hand, a Non Finite Automaton allows to model a system with an infinite number of states. A Deterministic Automaton corresponds to an Automaton where the transition between two states depends only on changes of internal variables of the system or the environment. On the other hand, a Non Deterministic Automaton is an Automaton in which several states can be reached from the current state. The choice between these multiple transitions can be done randomly. Most computer systems can be modeled with a Finite Automaton, whether deterministic or stochastic. A Deterministic Finite Automaton M can be defined with a tuple $(Q, \sum, \delta, q_0, F)$ such that:

- Q the set of states of the system;
- $\sum$ the set of symbols associated with transitions, either triggers or the result of the transition;
- δ the transition function such as $\delta : Q \times \sum \rightarrow Q$;
- $q_0 \in Q$ the initial state of the Automaton;
- $F \subseteq Q$ the set of final states of the Automaton.

Figure 41 shows a Deterministic Finite Automaton. The different states of the system are

represented by circles. The final states are represented by circles with two rings. The initial state is represented by an arc without a parent node. The transitions are represented by arcs between the nodes of the Automaton.

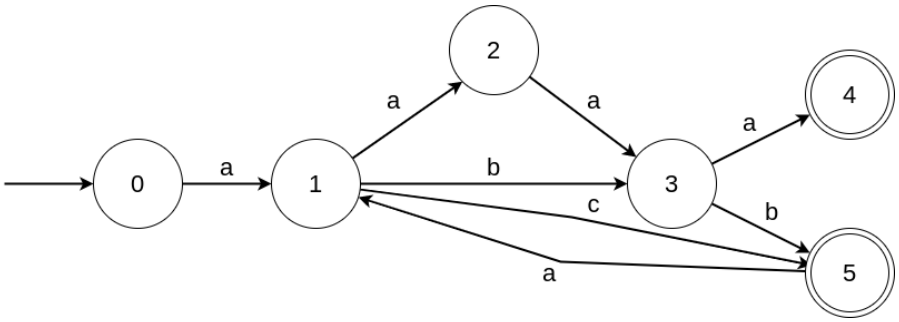


Figure 41: Example of a Deterministic Finite Automaton

Figure 42 represents a Non Deterministic Finite Automaton. We can see that unlike a Deterministic Finite Automaton, it is possible to have several transitions between two states of the system and with the same associated symbols. The choice between these transitions is made randomly from the probabilities indicated on the transitions. For example, it is possible to reach state S_2 from state S_1 with probability $P_{S_1 \xrightarrow{a} S_2} = 0.7$, and state S_3 with probability $P_{S_1 \xrightarrow{a} S_3} = 0.3$. The formal definition of a Non Deterministic Finite Automaton is very similar to that of a Deterministic Finite Automaton except for the definition of the transition function δ which becomes $\delta : Q \times \sum \rightarrow P(Q)$, with $P(Q)$ the power set of Q.

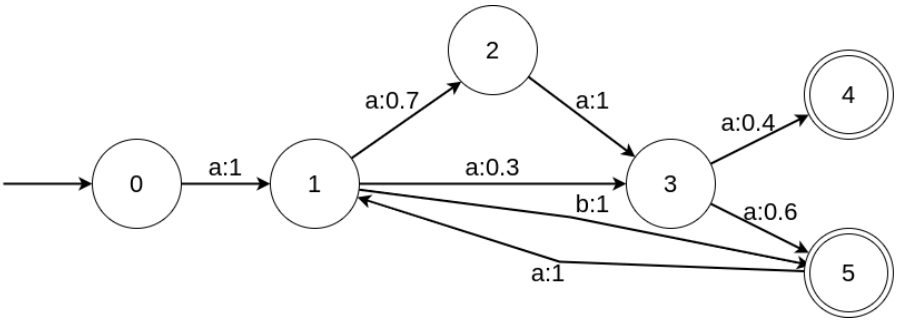


Figure 42: Example of a Non Deterministic Finite Automaton

Once the Finite Automaton is built, it is possible to check some properties with a model checking algorithm. A model checking algorithm explores the possible executions of the Automaton to check if a property P is satisfied. A property P is represented with a logical formula, most often with temporal logic [88].

The following section presents an example of using Deterministic Finite Automata to analyze the risk of compromise of a naval system. The last section shows why state-based models cannot

be used in the context of complex systems.

I.6.2 Security assessment of a naval system

In [42], the authors model a naval system composed of elements that can be affected by vulnerabilities. The exploitation of these vulnerabilities can impact the behaviour of the system. It is therefore necessary to apply patches to reduce the risk, without negatively impacting the functionality of the system. Timed Automata are used to model the state sequences of the behaviour of system components and missions, and quantitatively assess the impact of vulnerabilities, attacks, and countermeasures on naval missions.

Timed Automata are an extension of Finite Automaton where clocks can be defined. The values of these clocks all increase at the same speed during the execution of the Automaton. It is then possible to associate guards to the transitions of the Automaton which compare the value of the clocks to integers. With these clocks, timed Automata can model the temporal aspects of a system.

Dependencies between system and mission elements are modelled with guards assigned to the arcs between two states of the Automata. For example, the guard *movement_done?* between the states *shipMoving* and *measurementCampaign* ensures that the rudder operates correctly during the movement. Two types of Automata are used:

- system Automata which allow the behaviour of the vessel to be modelled. Timers allow to model the time needed for the operation of a component;
- the mission Automata make it possible to describe the different stages of a mission.

A set of properties are defined. These properties are used to verify the correct operation of the system using model checking, and are divided into three groups:

- A: the system can perform its tasks;
- T: the system can accomplish its tasks in a given time;
- I: the system can perform its tasks while maintaining the integrity of its components.

For any cyber event, whether it is the discovery of a vulnerability, an attack or a patch, changes to the system are modelled by making mutations to the system Automaton, such as

adding or removing a state, transition, clock, guard, or changing the value of a guard or clock. It is then possible for each mission to gather the modified system Automata with the mission Automaton, and to measure the impact on the system by checking different properties in the model. The construction of the Automata is the result of the federation of different models such as naval doctrines, missions, physical and functional architectures. A methodology for impact assessment is proposed:

1. definition of a nominal system Automaton A^n representing the initial state of the system before the discovery of a vulnerability, and satisfying the set of properties P;
2. definition of an Automaton of the vulnerable system A^v where mutations have been made from the Automaton A^n in order to describe the changes in the behaviour of the system after the discovery of a set of vulnerabilities. The matrix I^v represents the impact of the vulnerabilities on the satisfaction of the properties P;
3. for each countermeasure C_i, $i \in [1;l]$, an Automaton of the patched system A_i^p is constructed by performing a series of mutations from the Automaton A^v to model the effect of the patch C_i. The matrix I_i^p measures the impact of the patch C_i on the satisfaction of the properties P of the system. The difference between the matrices I^v and I_i^p measures the non-regression test of the countermeasure C_i;
4. for each attack att_j, $j \in [1;k]$, the Automata of the vulnerable system under attack $A_j^{v/att}$ and the Automata of the patched system under attack $A_{i,j}^{v/att}$ are constructed. The $I_j^{v/att}$ matrices measure the impact of the attack att_j on the vulnerable system, while the matrices $I_{i,j}^{p/att}$ measure the impact of the attack att_j on the system where the countermeasure C_i has been applied. The difference between the matrices $I_j^{v/att}$ and $I_{i,j}^{p/att}$ measures the effectiveness of the countermeasure C_i.

Mission 1	**Mission 2**
The state *end* of mission 1 Automata can be reached	The state *end* of mission 2 Automata can be reached
The state *end* of mission 1 Automata can be reached within x seconds	The state *end* of mission 2 Automata can be reached within x seconds
The rudder is operational	The network switch is operational
The controller is operational	The controller is operational
The network switch is operational	The rudder is operational

Table 10: List of properties to check for each mission Automaton

The model is tested on a small system composed of a rudder, its controller and a network switch allowing Internet access by the crew. Two missions are modelled (see Table 10): the

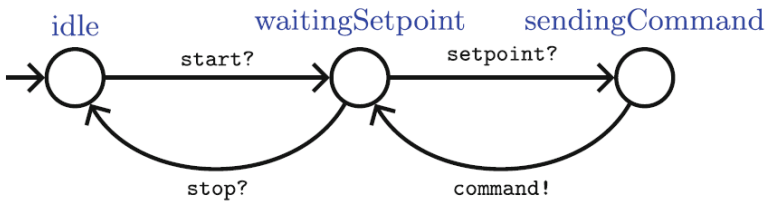


Figure 43: Automaton representing the behaviour of the rudder [42]

first is a measurement campaign (see Figure 44), and the second consists of transmitting the results of this campaign via the Internet. The network switch is modelled by an Automaton where a variable *internet_link* is set to 1 when it is connected to the Internet. The nominal Automaton A^n is the composition of the Automata present in Figure 44, 45 and 43, as well as the Automaton of the network switch. The Automaton A^n verifies the following properties:

- the end state of the PLC of the first mission can be reached;
- the end state of the Automaton of the first mission can be reached in x seconds;
- the end state of the PLC of the second mission can be reached;
- the end state of the Automaton of the second mission can be reached in x seconds;
- the rudder is operational;
- the controller is operational;
- the network switch is operational.

The controller is affected by a remotely exploitable DoS vulnerability, represented by the Automaton visible in Figure 46. The Automaton A^v corresponds to the composition of the Automata visible in Figure 44, 45 and 46, as well as the Automaton of the network switch. Two countermeasures C_1 and C_2 are defined, the first one consists in patching the controller software, the second one in forbidding the Internet connection to the network switch. I_1^p and I_2^p represent respectively the impact of the countermeasures on the Automata of the patched system A_1^p and A_2^p.

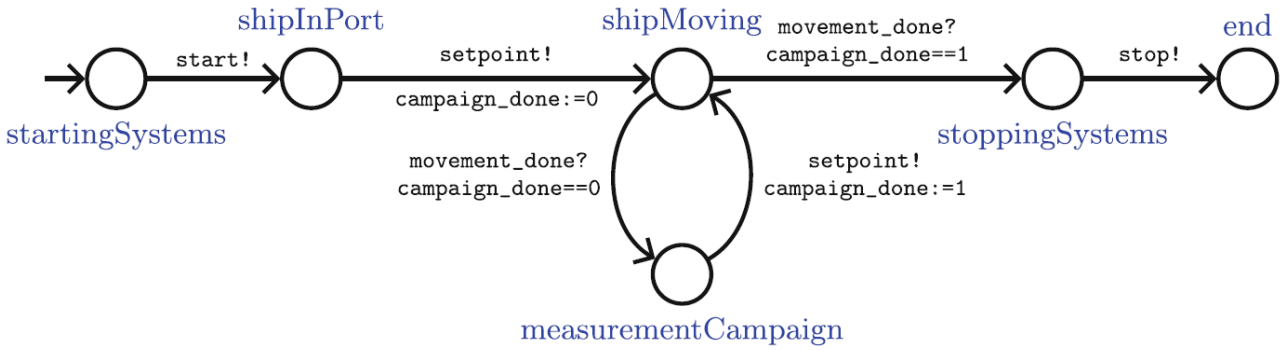


Figure 44: Automaton representing a measurement campaign [42]

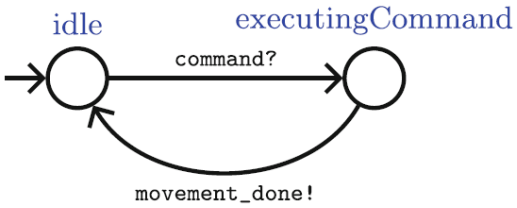


Figure 45: Automaton representing the behaviour of the rudder controller [42]

The Automaton in Figure 47 represents the attack att_1 which consists in exploiting the DoS vulnerability. The composition of the Automaton A^v and the Automaton visible in Figure 47 forms the Automaton $A_1^{v/att}$, and the matrix $I_1^{v/att}$ represents the impact of the attack att_1 on the vulnerable system. Each of the Automata A_1^p and A_2^p are merged with the attack Automaton visible in Figure 47, and the Automata $A_{1,1}^{p/att}$ and $A_{2,1}^{p/att}$ are obtained. The matrices $I_{1,1}^{p/att}$ and $I_{2,1}^{p/att}$ are then obtained and correspond respectively to the impact of the attack att_1 on the system patched with the countermeasure C_1 and C_2.

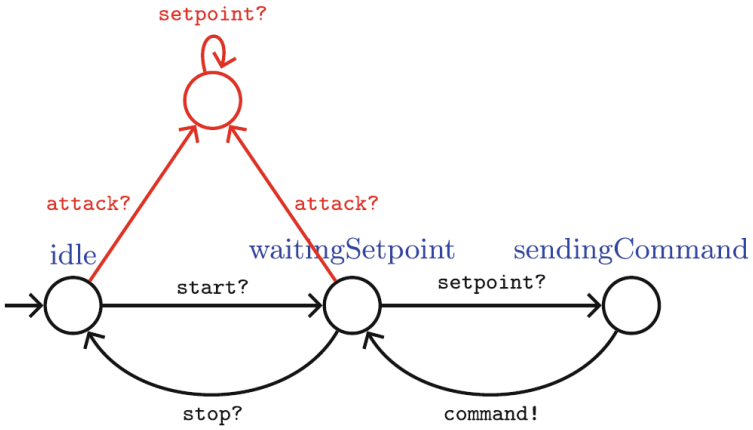


Figure 46: Automaton representing the behaviour of the rudder controller vulnerable to a DoS attack [42]

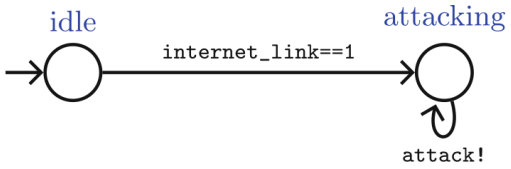


Figure 47: Automaton representing an attack on the rudder controller [42]

Each row of an impact matrix corresponds to a property and each column corresponds to a mission. The columns are sorted by the order of importance of the missions, while the elements of a column are sorted by their importance in the successful development of the corresponding mission. A lexicographical order, denoted $<_{lex}$ in the following, is defined to compare two impact matrices (see equation 45). The value of the different impact matrices corresponding to the test system can be seen in equation (46). The countermeasure C_2 impacts the proper functioning of the system because $I_1^p <_{lex} I^v$. The two countermeasures reduce the risk because $I_{1,1}^{p/att} >_{lex} I_1^{v/att}$ and $I_{2,1}^{p/att} >_{lex} I_2^{v/att}$.

$$\begin{aligned}&\text{Let } (a_{i,j}),(b_{i,j}) \in \{0,1\}^{m\times n}.\\&(a_{i,j}) <_{lex} (b_{i,j}) \iff \exists (k,l) \in [0,m] | (\forall p,q \in [0,m]\times\\&[0,l-1], a_{p,q} = b_{p,q}) \wedge (a_{0,l} = b_{0,l} \wedge ... \wedge a_{k-1,l}\\&= b_{k-1,l}) \wedge (a_{k,l} < b_{k,l}).\end{aligned} \tag{45}$$

$$\begin{aligned}I_v = \begin{bmatrix}1&1\\1&1\\1&1\\1&1\\1&1\end{bmatrix} I_1^p = \begin{bmatrix}1&1\\1&1\\1&1\\1&1\\1&1\end{bmatrix} I_2^p = \begin{bmatrix}1&0\\1&0\\1&0\\1&1\\0&1\end{bmatrix}\\ I_1^{v/att} = \begin{bmatrix}0&1\\0&1\\1&1\\0&0\\1&1\end{bmatrix} I_{1,1}^{p/att} = \begin{bmatrix}1&1\\1&1\\1&1\\1&1\\1&1\end{bmatrix} I_{2,1}^{p/att} = \begin{bmatrix}1&0\\1&0\\1&0\\1&1\\0&1\end{bmatrix}\end{aligned} \tag{46}$$

The complexity of the impact assessment depends on two parameters: the number of model checking performed for each vulnerability and the size of the Automata. The number of model

checking performed for each vulnerability is $n \times (m+1) \times (1+l+k+k \times l)$, where n is the number of missions, m the number of components, l the number of countermeasures and k the number of attacks. For a matrix with 20 missions, 2,000 components, 5 attacks per vulnerability and 5 patches, this requires 1,441,441 model checking. The first mission is composed of 39,069 states and 144,756 transitions, and the second mission of 4,095 states and 11,380 transitions.

I.6.3 Conclusion

Models based on the state of a system suffer from a problem of combinatorial explosion of the individual properties that compose it. For example, if we are interested in 6 properties in a system with each of them 10 different possible states, the state of the system is a combination of the values of these properties. The number of states of this system is therefore equal to 10^6. From a more general point of view, if we consider that a property of the system can only take 2 values, the number of possible states of a system is equal to 2^n, with n the number of properties to analyze. In complex systems composed of several thousands elements, there may be thousands of properties to analyze. The number of possible states of the system becomes too large, making its modeling very difficult and the exploration of its states with model checking algorithms impossible in a reasonable time. It is for this reason that we have decided to not explore further the solutions based on the modeling of the system states, such as Automata or Markov chains.

In the rest of this section, we will highlight the advantages and disadvantages of each solution and pose research questions that will define directions for our future work.

I.7 Discussion

This section presents the results of the literature review that was conducted on the problem of risk analysis in complex systems. This allowed us to assess the pertinence of the different models by discussing their advantages and disadvantages. Table 11 summarizes the evaluation of the main articles that have been presented in this chapter. As a reminder, the metrics used to compare articles have been defined in the introduction page 16.

We observe that all the solutions based on the modeling of attack paths allow to represent any type of impact. Indeed, it is possible to model the result of an attack with a node representing, for example, the unavailability, loss of confidentiality or integrity of a component. These models also make it possible to represent the multi-step attacks that a malicious actor can follow to

achieve its objectives, as well as the cascading effects that can be triggered. However, the generation of attack graphs is generally based on a static representation of the system. The only exception is the use of dynamic Bayesian networks in paper [64] to represent the evolution of the CVSS score of vulnerabilities, but this remains very limited.

The algorithms used to build attack graphs are adapted to the high complexity of the systems. However, an attack graph alone does not allow a very accurate assessment of the risk of compromise because it only lists the attack paths without taking into account other parameters such as the difficulty of exploiting the vulnerabilities.

Solutions have been proposed to perform a more accurate assessment of the risk of compromise based on attack graphs. In [36], the solution consists in listing all the attack paths present in an attack graph and then assessing their probability of success. But this poses problems when analyzing complex systems because the number of paths increases exponentially with the network size. In [43], [46] and [47], the proposed solutions allow to evaluate the probability of success of the attacker's actions without having to list all the attack paths and by considering the uncertainty of some information. However, the probability calculations performed in [46] were proved to be inaccurate and those performed in [47] are inadequate for large systems.

The use of Bayesian networks, such as those used in [58] and [64], allow to calculate the probabilities of success of the attacker's malicious actions. Nevertheless, the calculations performed are not adapted to complex systems, except by performing approximate calculations like those presented in [54].

Petri nets can be used to represent the attack paths and approximate the success probabilities as the solution proposed in paper [75]. Moreover, Petri nets can be used to model the uncertainty associated with some information.

Epidemiological models and complex network theory are suitable for risk analysis in large systems. However, these models are generally used respectively to represent the propagation of a single type of attack [78], [81] and to model the impact of the unavailability of some components on the proper functioning of the system [85], [86]. More work should be done to verify if these models can be adapted to model other types of impact such as loss of confidentiality or integrity of a component. Moreover, the complex network theory is a purely deterministic model and does not allow to model the uncertainty about the information related to the system.

This literature review has highlighted the most relevant solutions to analyze the security of complex systems. We observe a clear dominance of solutions based on the modeling of attack paths. Indeed, these models allow to represent very precisely the ways in which a malicious

agent can compromise the system.

Although algorithms for generating attack graphs capable of handling large systems have already been developed, the subsequent computations performed to calculate the risk of compromise are rarely adapted to complex systems. One may wonder if it is possible to propose a risk analysis solution based on attack graphs capable of handling large systems, while considering the uncertainty that may exist about some information.

Furthermore, attack graphs are generated from a static description of the system and therefore cannot take into account changes that may occur during its life cycle, such as variations in network topology. Thus, it should be investigated whether it is possible to generate an attack graph from a system model that includes its dynamic behavior.

Finally, we could investigate if solutions exist to simplify the system modeling step required for the generation of the attack graph, since this can quickly become cumbersome for systems composed of several thousands of elements.

Article	Impact Assessment - IA	System Modeling					Complexity	
		Cascading Effects - CE	Multi-step Attacks - MA	Dynamic Properties - DP	Stochastic Properties - SP	Uncertainty - UN	Modeling Complexity - MC	Risk Assessment Complexity - RAC
Multi-step attack modeling								
Attack graph generation								
[24]	$IA > 4$	U	U	N	N	N	$MC \leq 100$	$RAC \leq 100$
[25]	$IA > 4$	U	U	N	L	N	$MC \leq 100$	$RAC \leq 100$
[26], [28], [29], [89]	$IA > 4$	U	U	N	N	N	$MC > 1000$	$RAC > 1000$
[31], [34], [45]	$IA > 4$	U	U	N	L	N	$MC > 1000$	$RAC > 1000$
[35]	$IA > 4$	U	U	N	N	N	$MC > 1000$	$RAC > 1000$
Attack graph based risk assessment								
[36]	$IA > 4$	U	U	N	L	N	$MC > 1000$	$100 < RAC < 1000$
[41]	$IA > 4$	U	U	N	N	N	$MC > 1000$	$100 < RAC < 1000$
[43]	$IA > 4$	U	U	N	U	L	$MC > 1000$	$RAC > 1000$
[46]	$IA > 4$	U	U	N	L	L	$MC > 1000$	$RAC > 1000$
[47]	$IA > 4$	U	U	N	U	L	$MC > 1000$	$RAC \leq 100$
Bayesian Network								
[58]	$IA > 4$	U	U	N	U	L	$MC \leq 100$	$RAC \leq 100$
[64]	$IA > 4$	U	U	L	U	L	$MC > 1000$	$RAC \leq 100$

Petri net								
[74]	$IA > 4$	U	U	N	N	N	$MC > 1000$	$RAC \leq 100$
[75]	$IA > 4$	U	U	N	U	L	$MC > 1000$	$RAC > 1000$
Epidemiological models								
[78]	$IA \leq 1$	L	L	L	N	N	$MC > 1000$	$RAC > 1000$
[81]	$IA \leq 1$	U	U	L	U	L	$MC > 1000$	$RAC > 1000$
Complex network theory								
[85]	$IA \leq 1$	L	L	L	N	N	$MC > 1000$	$RAC > 1000$
[86]	$IA \leq 1$	L	L	L	N	N	$MC > 1000$	$RAC > 1000$
State-based models								
[42]	$IA > 4$	U	U	U	N	N	$MC \leq 100$	$RAC \leq 100$

Table 11: Evaluation of the most relevant articles

I.8 Conclusion

In this literature review, we have defined several metrics in order to evaluate the relevance of the different solutions in solving our research problem. We were able to highlight that solutions based on the modeling of attack paths were the most adapted to assess the security in complex systems. We then identified some limitations to the current solutions that allowed us to formulate several research questions. In the following chapter, we will present the work done during this thesis to answer the research questions previously defined.

Chapter II

Dynamic security assessment of complex systems

II.1 Introduction

The literature review conducted in the previous section has identified several limitations to current risk analysis solutions used in complex systems. The objective of this Ph.D thesis is to propose a solution allowing to remove all the obstacles that we identified, which are:

1. model the attack paths from a system model including its dynamic properties such as network topology changes. The generation of the attack graph must be achievable in a reasonable time even for systems composed of several thousands elements.

2. assess the risk of the system being compromised by a malicious actor from the attack graph previously constructed. This evaluation must take into account the difficulty of the actions performed by the attacker as well as the uncertainty about the real state of the system and the real capabilities of the attacker. Moreover, this evaluation must be achievable in a reasonable amount of time even when the size of the attack graph reaches several million of nodes.

3. facilitate the system modeling step required to build the attack graph.

To achieve these goals, we started by looking for a complex system from which we could test our model. Unfortunately, we did not find any system architecture complex enough and publicly available. So we decided to create our own dataset representing the architecture of an IT system of a major organization. To ensure that our dataset is representative of a real IT system, we validated our model by several domain experts. Then, after having developed our solution theoretically, we implemented it in a tool called *DAGSIM*. This tool allowed us to test our solution in real conditions by applying it to our dataset representing the IT network of a major organization.

In the rest of this chapter, we will present our attack graph model based on a dynamic representation of the system, as well as our method for assessing the risk of compromise. We will also illustrate the tests that have been performed to ensure that our solution can be used on complex systems.

II.2 Presentation of the dynamic attack graph model

II.2.1 Presentation of the use case

We have created a use case to illustrate our risk analysis method based on dynamic attack graphs. We have chosen to model a user who is regularly working from home in his company. Indeed, this way of working has developed rapidly in recent years and we want to analyze the impact of this practice on the security of the computer system. The system consists of a mobile user station and a web server. A firewall restricts the access to the company's computer system and only users connected to the internal network can access the web server. An attacker has already compromised the user's home WiFi network. **We aimed to model how this attacker could access the company's internal network and compromise the web server over a period of three weeks**. The user is on the company's internal network the first week, then connects to its home network the second week and returns to the internal network the third week. The vulnerability CVE-2020-0796 is present on the SMB (Server Message Block) service of the user workstation which is only accessible from the local network. This vulnerability allows the attacker to execute code on the machine and gain remote control. A second vulnerability, the CVE-2017-12617, is present on the server's web service that is only accessible from the internal network. This vulnerability allows the attacker to execute arbitrary code on the server. The network is visible in Figure 48 and the characteristics of the vulnerabilities are summarized in Table 12. The CVSS metrics presented in this table are then used to calculate the mean time required to exploit them.

There are two attack paths in this use case: the first one allows the attacker to compromise the user station when it is connected on the home WiFi network the second week. At this moment, the SMB service is accessible because the attacker has already compromised the WiFi network and therefore has access to the local network. Once the vulnerability on the SMB service is exploited, the malicious actor is able to execute arbitrary code. A Meterpreter is then used to obtain a remote access on the user station. A second attack path allows to execute arbitrary code on the web server. Once the user's computer is compromised, the attacker can wait the third week for the user to connect to the company's internal network again. At this moment, the attacker has access to the web server from the compromised machine and can exploit the vulnerability present on the web service. The particularity of this attack path is that it requires the user station to move and change network. **This use case illustrates the importance of modeling the dynamic behavior of systems, as new attack paths can be identified.** We will see how a dynamic system can be modeled with a dynamic attack graph.

CVE ID	Description	Attack Complexity (AC)	Privileges Required (PR)	User Interaction (UI)
CVE-2017-12617	Remote code execution on Apache Tomcat	High (0.44)	None (0.85)	None (0.85)
CVE-2020-0796	Remote code execution on SMBv3	Low (0.77)	None (0.85)	None (0.85)

Table 12: Characteristics of the vulnerabilities present in the remote working use case network

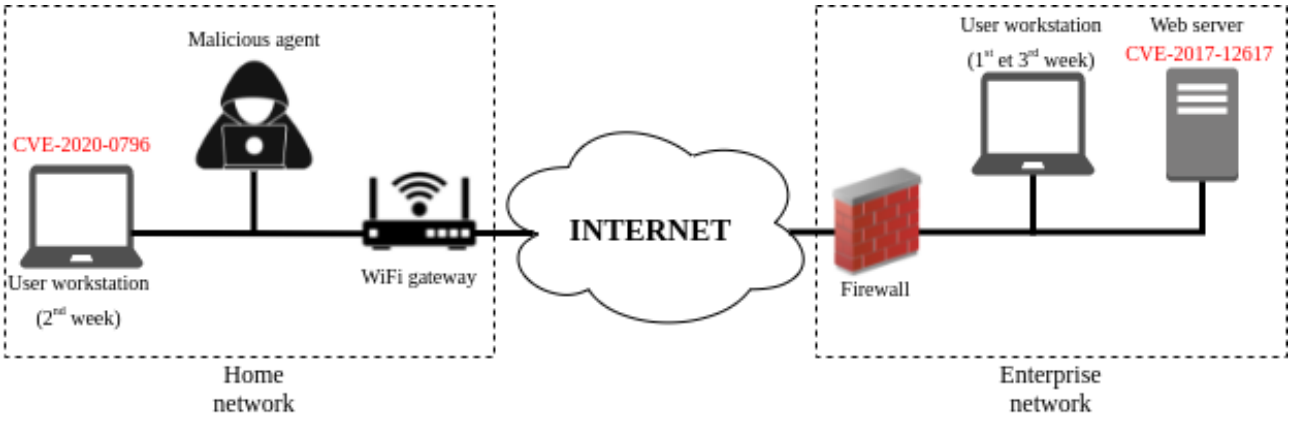


Figure 48: Remote working use case network

II.2.2 System modeling

To model the attack paths present in a system, we relied on the work done on logic-based attack graphs [31]. As a reminder, the MulVAL framework was presented in the survey section page 26. The properties of the system are represented with literals. A literal is a logical formula that can take a Boolean value True or False, and it is composed of a predicate followed by its parameters. A predicate represents a type of information such as the presence of a vulnerability or a network connection. The parameters allow this information to be contextualized with respect to the system being modeled. For example, the parameters of the predicate vulExists indicate the name of the vulnerability and its impact on the system.

Definition 1 *A literal L is defined as a predicate P applied to its parameters $(x_1, \cdots, x_n)$ such that $L := P(x_1, \cdots, x_n)$. A literal set LS_j is the set of literals formed with the same predicate P_j such that $L := P_j(x_1, \cdots, x_n)$.*

For example the vulnerability CVE-2017-12617 present on the web service can be modeled with the predicate *vulExists* followed by its parameters:

vulExists(webServer, 'CVE-2017-12617', httpd, remoteExploit, privEscalation)

This literal indicates that the vulnerability CVE-2017-12617 present on the httpd service of the web server can be exploited remotely and allows an attacker to execute code with the privileges of the service. Several predicates are predefined in the MulVAL framework. We decided to define new predicates in our tool to model computer systems more accurately. The complete list of the predicates are defined in Table 14 in the appendices.

To model the changes of states of the system, we assign time intervals $IS = \{I_1, I_2, \cdots, I_m\}$ to each literal. Each time interval $I_i = (t_i, t_i', p_i)$ assigned to a literal defines a time period during which it is equal to the Boolean value True with probability p_i. These literals are called temporal literals.

Definition 2 *A temporal literal L is defined as a predicate P applied to its parameters $(x_1, \cdots, x_n)$ and on a set of time intervals IS such that $L \coloneqq P(x_1, \cdots, x_n, IS)$. A temporal literal set LS_j is the set of temporal literals formed with the same predicate P_j such that $L \coloneqq P_j(x_1, \cdots, x_n)$.*

These temporal literals are used to model network topology changes, time periods between the discovery of a vulnerability and the application of a patch, as well as the wipe of a computer component. The probabilities associated with the time intervals allow to model the uncertainty that we may have about an information. For example, if we are not totally sure that the vulnerability *CVE-2017-12617* is present on the web server, we can set the probability p to 0.8.

Changes in the network connections of a computer device can be modeled with the temporal literal *vlanInterface*. For example, in the home working use case, the network change of the computer station during the three weeks can be modeled with the following two temporal literals:

(1) vlanInterface(workstation, userLAN).[(2021-01-04 00:00:00, 2021-01-10 23:59:59, 1), (2021-01-18 00:00:00, 2021-01-24 23:59:59, 1)]

(2) vlanInterface(workstation, homeNetwork).[(2021-01-11 00:00:00, 2021-01-17 23:59:59, 1)]

The first literal indicates that the user's computer is connected to the company network from 2021-01-04 00:00:00 to 2021-01-10 23:59:59 with probability 1, from 2021-01-11 00:00:00 to 2021-01-17 23:59:59 with probability 0 and from 2021-01-18 00:00:00 to 2021-01-24 23:59:59 with probability 1. The second literal indicates that the user station is connected to the home network from 2021-01-04 00:00:00 to 2021-01-10 23:59:59 with probability 0, from 2021-01-11 00:00:00 to 2021-01-17 23:59:59 with probability 1 and from 2021-01-18 00:00:00 to 2021-01-24 23:59:59 with probability 0.

The time between the publication of a vulnerability and the application of a patch can be modeled with the temporal literal *vulExists*. For example in our use case, the vulnerability CVE-2020-0796 was published on 2020-12-03 00:00:00 and a patch was applied on 2021-02-01 12:00:00. This information can be modeled with the following temporal literal:

vulExists(workstation, 'CVE-2020-0796', smb, remoteExploit, privEscalation).[(2020-12-03 00:00:00, 2021-02-01 12:00:00, 1)]

This temporal literal indicates that the CVE-2020-0796 vulnerability is only present on the SMB service of the user station from 2020-12-03 00:00:00 to 2021-02-01 12:00:00 with probability 1 and with probability 0 otherwise.

The wipe of a computer component has the consequence of cancelling the compromises previously made by the attacker. This can be modeled with the temporal literals execCode, availability, DDoS (Distributed Denial of Service) and confidentiality. For example in our use case, it is possible to model the wipe of the computer with the following literals:

execCode(workstation, root).[(2021-01-22 00:00:00, 2021-01-22 23:59:59, 0)]

availability(workstation, smb).[(2021-01-22 00:00:00, 2021-01-22 23:59:59, 0)]

ddos(workstation, root).[(2021-01-22 00:00:00, 2021-01-22 23:59:59, 0)]

confidentiality(workstation_data).[(2021-01-22 00:00:00, 2021-01-22 23:59:59, 0)]

These literals indicate that the workstation wipe started on 2021-01-22 at 00:00:00 and ended on 2021-01-22 at 23:59:59. The wipe of a component can be decided by the security teams in case for example a compromise has been detected.

The interactions that take place in the system are modeled with Horn clauses [90] and are written in the Datalog environment [32]. A Horn clause is a logical formula written as a rule. They allow to model reasoning rules and to deduce new information about the system from already known facts.

Definition 3 *A reasoning rule R is a relation that allows to deduce a new literal L_R from the product of literal sets $LS = LS_1 \times \cdots \times LS_n$ such that $R\colon LS \mapsto \{L_R, NULL\}$.*

It is possible to model the exploitation of a vulnerability by an attacker with the following reasoning rule:

execCode(H, Perm) :-
vulExists(H, _, Software, remoteExploit, privEscalation),
networkServiceInfo(H, _, Software, Protocol, Port, Perm),
netAccess(H, Protocol, Port)

This reasoning rule indicates that if a remotely exploitable elevation of privilege vulnerability is present on a service of a computer component, that this service is running and communicating on the network, and that the attacker has access to the listening port with the right protocol, then he can exploit it and execute code on the machine with the service rights. In other words, if the literals *vulExists*, *networkServiceInfo* and *netAccess* exist in the system with the correct parameters, then a new literal *execCode* is created. In the reasoning rules, the parameters that start with a capital letter are variables that allow to logically link the different literals. For example the variable Software must contain the same value in the literal vulExists and networkServiceInfo for the literal execCode to be created. The presence of an underscore means that any value can be present in this literal parameter. The list of reasoning rules predefined in MulVAL can be found in [31]. We have defined other rules to model new interactions in the computer system and they are presented in Table 15 given in the Appendix.

We assign clocks to the reasoning rules in order to model the time required for the realization of interactions in the system. A clock indicates the time during which the set of preconditions must remain True in a reasoning rule for the new literal to be created. We call these rules temporal reasoning rules. In the remote working use case, to model the time t_e required to exploit the vulnerability CVE-2020-0796, we associate a clock $h = t_e$ to the reasoning rule *execCode* when the parameter *vulID* of the literal *VulExists* is equal to 'CVE-2020-0796'. But

the changes of states of the system can change the time required to perform some interactions in the system. For example, the difficulty of exploiting the vulnerability CVE-2020-0796 was initially considered important because there was no publicly available exploit code. But on June 29, 2020, a PoC (Proof of Concept) was published on a GitHub, making the attacker's job much easier. To model these changes, several clocks can be defined on different time intervals. For example, for the exploitation of the vulnerability CVE-2020-0796, a first clock h_1 can be assigned from March 12, 2020, date of its publication, to June 29, 2020, date of the POC publication. Then a second clock h_2 from June 29, 2020 until today, with $h_1 >> h_2$. More formally, a clock set $CS = \{C_1, \cdots, C_n\}$ is assigned to a reasoning rule R, with $C_i = (t_i, t_i', colck_i)$ a clock of duration $clock_i$ valid from time t_i to time t_i'.

Definition 4 *A temporal reasoning rule R is a relation that allows to deduce a new temporal literal L_R from the product of a temporal literal set LS and a clock set CS such that R: $LS \times CS \mapsto \{L_R, NULL\}$, and $R(L_1, \cdots, L_n, clock_i) = L_R \Rightarrow clock_i \leq 0 \mid (t_i, t_i', clock_i) \in CS$.*

The value $clock_i$ of a clock $C_i \in CS$ may in some cases not be known exactly. For example, the time it takes for the attacker to exploit a vulnerability depends on a large number of factors unknown to the defender, such as the level of the attacker or the resources deployed. To take this uncertainty into consideration, we have decided to represent the value of a clock with a probability distribution.

In [91], the authors look for the probability distribution that best fits the time spent by an attacker to compromise a system. This analysis is based on intrusions detected on more than 260,000 computer systems. The lognormal distribution with parameters $\mu = 4.005$ and $\sigma = 1.247$ is the one that best fits the observed data. We therefore used this probability distribution to model the time required for an attacker to exploit a vulnerability in the attack graph.

We propose a solution to compute the parameters μ and σ of the log normal distribution of a clock associated with a vulnerability exploitation. First, we compute a score that measures the difficulty of exploiting the vulnerability from the CVSS metrics. The CVSS metrics of a vulnerability can be found in databases like NVD. There are two versions of the CVSS score. In the case of version 2, we use the metrics *Access Complexity* and *Authentication* to compute our normalized exploitation score $exploit\_score \in [0, 1]$ with the following equation:

$$\begin{aligned} exploit\_score &= \frac{-1}{0.34234} \times x + (1 + \frac{0.1575}{0.34234}) \\ \text{with } x &= access\_complexity \times authentication \end{aligned} \tag{47}$$

The metric *Access Complexity* considers the privileges required to exploit the vulnerability, the additional actions to be performed or the interactions required with the victim. The metric *Authentication* indicates whether the attacker must authenticate to exploit the vulnerability. We do not consider the metric *Access Vector* which indicates the way of access to the vulnerable service, because this information is encoded in the structure of the attack graph. In version 3 of the CVSS score, we use the metrics *Attack Complexity*, *Privileges Required*, and *User Interaction* to calculate our normalized exploitation score $exploit\_score \in [0,1]$ with the following equation:

$$\begin{aligned} exploit\_score &= \frac{-1}{0.482669} \times x + (1 + \frac{0.073656}{0.482669}) \\ \text{with } x &= attack\_complexity \times privileges\_required \times user\_interaction \end{aligned} \tag{48}$$

The metric *Attack Complexity* takes into account the conditions required to exploit the vulnerability that are not under the control of the attacker. The metric *Privileges Required* measures the level of privileges required by the attacker. The metric *User Interaction* indicates whether a user must perform some actions for the attacker to exploit the vulnerability. As with version 2 of the CVSS score, we do not consider the metric *Attack Vector* because this information is contained in the structure of the attack graph. The choice between the equation (47) and (48) depends on the availability of the version of the CVSS score associated with the vulnerability. The equation (47) should only be used if version 3 of CVSS score is unavailable.

The objective is now to calculate the mean time M required for an attacker to exploit a vulnerability from its previously calculated exploitation score *exploit_score*. To do this, we have to make several assumptions: (1) the time required to exploit a vulnerability is at least 3,600 seconds for an easily exploitable vulnerability, and at most 2,678,400 seconds or 1 month for a vulnerability that is very difficult to exploit. (2) The mean exploitation time of a vulnerability evolves linearly with the exploitation score *exploit_score*. We can now calculate the mean exploitation time of a vulnerability from the exploitation score *exploit_score* with equation (49).

$$M = 2678400 \times exploit\_score + 3600 \tag{49}$$

Now that we have calculated the mean exploitation time of a vulnerability, we need to define a value for the standard deviation *SD* in order to model the uncertainty. As a reminder, in [91], the time required to compromise a system is modeled by a lognormal distribution of parameters $\mu = 4.005$ and $\sigma = 1.247$. From these parameters, we calculated with equation 50 and 51 the mean value *M* of the data as well as the standard deviation *SD*, respectively equal to 119 and 231. As a reminder, the variance is the square of the standard deviation. The standard deviation is equal to 200% of the mean value of the observed data. We therefore used this percentage of uncertainty to calculate the standard deviation associated with the mean exploitation time of a vulnerability.

$$M = exp(\mu + \frac{\sigma^2}{2}) \tag{50}$$

$$V = (exp(\sigma^2) - 1) \times exp(2 \times \mu + \sigma^2) \tag{51}$$

It is possible to calculate the parameters μ and σ of the log normal distribution with the mean and variance of the random variable with equations (52) and (53). The calculations that led to these equations are detailed in demonstration (1). In our tool, we automated the calculation of the parameters μ and σ from the data stored in the NVD database.

$$\sigma^2 = ln(\frac{V}{exp(2 \times ln(M))} + 1) \tag{52}$$

$$\mu = \frac{ln(\frac{V}{exp(\sigma^2)-1}) - \sigma^2}{2} \tag{53}$$

We also propose to model the learning of the attacker during its infiltration in the system. To do this we decided to reduce the mean time to exploit a vulnerability by 40% if the attacker has already exploited it in the past. For example, the parameters of the normal log distribution representing the exploitation time of the vulnerability CVE-2020-0796 are initially $\mu = 7.38397$ and $\sigma = 1.26864$. But if the attacker exploits this vulnerability a second time, the parameters are recalculated taking into account the 40% decrease of the mean exploitation time and we obtain $\mu = 6.4308$ and $\sigma = 1.57928$.

The nodes in the dynamic attack graph representing a literal deduced by a reasoning rule can have a memory. This memory indicates that if the literal is true at time t, then it will remain True for all times $t' > t$. This makes it possible to model phenomena that persist in time even when the initial preconditions are no longer satisfied. For example, we have assigned a memory to the literal *execCode* because when an attacker manages to meet the necessary conditions to exploit a vulnerability and execute code on a machine, this code continues to run even if the vulnerability is patched. The only way for the code to stop running is for security teams to detect the attack and decide to stop the execution of the malicious program.

Demonstration 1

Let V be the variance of a log normal distribution [92]

$$
\begin{aligned}
&(exp(\sigma^2)-1)\times exp(2\times\mu+\sigma^2)=V\\
&\implies exp(2\times\mu+\sigma^2)=\frac{V}{exp(\sigma^2)-1}\\
&\implies 2\times\mu+\sigma^2=ln(\frac{V}{exp(\sigma^2)-1})\\
&\implies 2\times\mu=ln(\frac{V}{exp(\sigma^2)-1})-\sigma^2\\
&\implies \mu=\frac{ln(\frac{V}{exp(\sigma^2)-1})-\sigma^2}{2}
\end{aligned}
$$

Let M be the mean of a log normal distribution [92]

$$
\begin{aligned}
&exp(\mu+\frac{\sigma^2}{2})=M\\
&\implies \mu+\frac{\sigma^2}{2}=ln(M)\\
&\implies \frac{ln(\frac{V}{exp(\sigma^2)-1})-\sigma^2}{2}+\frac{\sigma^2}{2}=ln(M)\\
&\implies ln(\frac{V}{exp(\sigma^2)-1})-\sigma^2+\sigma^2=2\times ln(M)\\
&\implies ln(\frac{V}{exp(\sigma^2)-1})=2\times ln(M)\\
&\implies \frac{V}{exp(\sigma^2)-1}=exp(2\times ln(M))\\
&\implies exp(\sigma^2)-1=\frac{V}{exp(2\times ln(M))}\\
&\implies exp(\sigma^2)=\frac{V}{exp(2\times ln(M))}+1\\
&\implies \sigma^2=ln(\frac{V}{exp(2\times ln(M))}+1)
\end{aligned}
\tag{54}
$$

Definition 5 *A literal L with a memory is defined as $Eval(L,t_i)=TRUE \implies \forall j\in[i+1,+\infty]$, $Eval(L,t_j)=TRUE$.*

Now that the system modeling step has been explained, the construction of the dynamic attack graph will be detailed in the next section.

II.2.3 Attack graph generation

We rely on the work done in paper [31] to build our dynamic attack graph. This framework uses the Prolog system XSB to recursively derive new literals from the initial literals and reasoning rules. This succession of deduction of new literals is represented in the form of a graph which is called logic-based attack graph. The initial literals are represented by rectangles and are the root nodes of the graph. The reasoning rules are represented by ovals. Their parent nodes are literals, initial or inferred, and their child node is an inferred literal. The inferred literals are represented with diamonds. The advantage of this attack graph construction solution is that it allows to model the attack paths present in any type of system. Indeed, this framework offers the possibility to define literals and reasoning rules adapted to the properties of the studied system. The construction of the dynamic attack graph works as follows:

1. our algorithm retrieves the temporal literals and temporal reasoning rules that the user has defined to model the system.
2. to generate the attack graph with the MulVAL framework, we must temporarily remove the time intervals associated with the literals and the clocks associated with the reasoning rules.
3. the MulVAL framework is then used to generate an attack graph. Therefore, the resulting graph contains more attack paths than actually exist in the system since we temporarily consider that the literals are true all the time and that the reasoning rules are instantaneous.
4. the time intervals are then attributed again to the literals present in the graph, while the clocks are attributed again to the reasoning rules.

Figure 49 represents the dynamic attack graph that was constructed to represent the attack paths present in the remote working use case. The green rectangles represent the initial literals that are true all the time. The gradient rectangles indicate an evolution of the probability of the literal being true. The white ovals represent the instantaneous reasoning rules with a clock set to 0. The gradient ovals indicate that a delay is required to trigger the reasoning rule. The two colored rectangles in gradients correspond to the two literals *VlanInterface* which represent the connection changes of the user's computer. The two colored ovals in gradients correspond

to the two vulnerability exploits that the attacker can perform in the system. The clocks associated with these nodes follow the log-normal distribution visible on the figure. However, a dynamic attack graph does not allow easy visualization of the attack paths because of the temporal characteristics associated with the nodes. In the next section, we will show how this graph can be used to approximate the attacker's success probabilities.

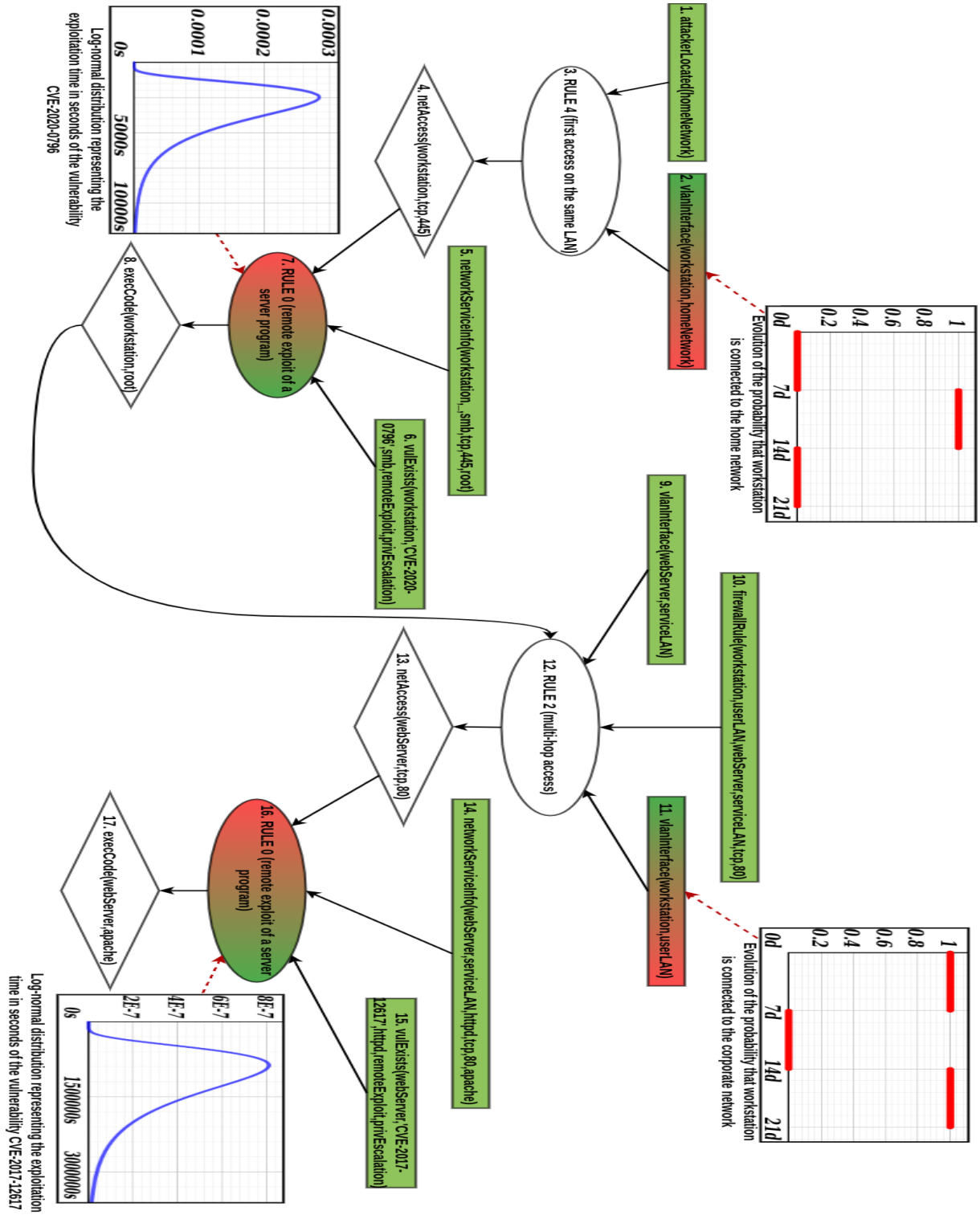


Figure 49: Dynamic attack graph for the remote working use case

II.3 Assessment of the risk of compromise based on dynamic attack graphs

II.3.1 Simulation of an attack

Our objective is to measure the risk to compromise the system from the dynamic attack graph built previously. To do this, we compute for each node of the graph its probability of being True as a function of time. For the nodes representing a literal, the value TRUE means that the property represented by the literal is true. For the nodes representing a reasoning rule, the value TRUE means that the reasoning rule is activated. We aim to approximate this probability by performing several simulations of an attacker progressing through the system. Each simulation allows to explore all the attack paths present in the graph. We have chosen to use an approximate solution because it allows to manage efficiently large systems as we will see in the next section. The different parameters required for this simulation are a start date t_s, an end date t_e and a progress step τ. The simulation step τ allows to define the frequency with which the attacker's progress is calculated. A smaller step value will result in a more accurate simulation but will also require more time to complete. The start and end date of the simulation depends on how long the attacker is willing to spend to compromise the system. An opportunistic attack can last several days while a cybercriminal group or a state is able to stay several months or even years. The chosen dates will also define the existing vulnerabilities at that time as well as potential releases of exploit code. We will now see how these simulations allow to approximate the probability for each node of the attack graph to be True as a function of time.

A matrix $M^2 = (a_{i,j}), i \in I, j \in J$ is defined such that $I = \{t \in \mathbb{N} | (t - t_s) \bmod \tau = 0\}$, $J = \{v \in \mathbb{N} | 1 \leq v \leq m\}$ with m the size of the graph and $a_{ij} \in \{0,1\}$. The rows of the matrix represent the different time slices t_i while the columns represent the nodes of the attack graph v_j. An element of the matrix a_{ij} is equal to 1 if the node v_j is True at time t_i and 0 otherwise. The aim of the simulation is to fill this matrix by traversing all the nodes v_j of the graph for each time slice t_i. In our simulation, we modeled an attacker with unlimited budget and human resources. There are several reasons for this choice: (1) we are sure to explore all the attack paths present in the graph and obtain a complete analysis of the system security. (2) The unlimited number of human resources allows to parallelize the exploration of the different attack paths during the simulation and significantly increases the performance of our algorithm. We therefore perform a worst-case security analysis.

Our algorithm works as follows:

1. an initialization phase at time $t_0 = t_s$ allows to assign a Boolean value to all the literals that have a probability p_n such that $(t_n, t'_n, p_n) \in I_n, I_n \in IS | L_j := p(x_1, \cdot, x_n, IS)$. A Boolean value True is assigned to the literal L_j with probability p_n and a value False with probability $q_n = 1 - p_n$. The clocks associated with the reasoning rules whose value is defined by a probability distribution are also initialized. To do this, we sample the probability distribution and assign the resulting value to the clock.

2. the attack graph is then traversed with a modified Depth-First Search (DFS) algorithm for each time slice t_i. A first loop in the algorithm goes through all the nodes that have been previously initialized. For each of these nodes, we look for the child nodes that can be calculated. This verification as well as the calculation of the node is done with the Algorithm 2 that we will detail further in this section. This algorithm recursively traverses the nodes of the attack graph to assign them a Boolean value True or False.

3. once the attack graph is traversed for a time slice t_i, the Boolean values assigned to all nodes v_j are written in row i of the matrix M^2 such that $a_{ij} = 1 \leftrightarrow v_j = True$, and 0 otherwise. Once the graph is traversed for each time slice t_i, the matrix M^2 is completed.

Algorithm 2 Function ComputeNode

Input: $v \in NODES, t \in T$

if $(v,t_i,t_i',bool_i) \in I_i \mid t \in [t_i,t_i']$ **then**
  if $bool_i = true$ **then**
    $TRUE = TRUE \cup (v,t)$
  else
    $FALSE = FALSE \cup (v,t)$
  end if
else if $(v,t-\tau) \in (TRUE \cap STATE)$ **then**
  $TRUE = TRUE \cup (v,t)$
else if $v \in LITERALS$ **then**
  if $\exists (p,v) \in E \mid (p,t) \in TRUE$ **then**
    $TRUE = TRUE \cup (v,t)$
  else
    $FALSE = FALSE \cup (v,t)$
  end if
else if $v \in RULES$ **then**
  if $\forall (p,v) \in E \mid (p,t) \in TRUE$ **then**
    $clock_i = clock_i - \Delta_t \mid (v,d_i,d_i',clock_i) \in C_i \wedge t \in [d_i,d_i']$
    if $clock_i \leq 0$ **then**
      $TRUE = TRUE \cup (v,t)$
      REINIT($clock_i$)
    else
      $FALSE = FALSE \cup (v,t)$
    end if
  else
    $TRUE = TRUE \cup (v,t)$
  end if
else
  $FALSE = FALSE \cup (v,t)$
end if
$COMPUTED = COMPUTED \cup (v,t)$
for $(v,child) \in E$ **do**
  if $(child,t) \notin COMPUTED \wedge \forall (p,child) \in E \mid (p,t) \in TRUE$ **then**
    ComputeNode($(child,t)$)
  else if $child \in LITERALS$ **then**
    if $(child,t) \notin COMPUTED \wedge \exists (p,child) \in E \mid (p,t) \in TRUE$ **then**
      ComputeNode($(child,t)$)
    end if
  end if
end for

Algorithm 2 recursively traverses the nodes of the attack graph for each time slice t_i to assign a Boolean value True or False. The algorithm works as follows:

1. the first line allows to check if the node v_j given in parameter is a literal with a Boolean value that has been assigned during the initialization phase. If so, the value is retained.
2. line 7 checks whether the node v_j is a literal with a memory and with a Boolean value True assigned at time t_{i-1}. In this case, a Boolean value True is assigned to the node.
3. line 9 checks whether the node v_j is a literal. If there is a parent node of v_j with a Boolean value True assigned at time t_i, then a Boolean value True is assigned to node v_j. Otherwise, a Boolean value False is assigned.
4. line 15 checks if the node v_j is a reasoning rule. If all parent nodes of v_j have a Boolean value True assigned at time t_i, then the clock associated with node v_j is decremented by the duration of one simulation step τ. If the final value of this clock is less than 0, then a Boolean value True is assigned to node v_j and the clock value is reset. If the value of the clock remains greater than 0 or if one of the parent nodes of v_j is not True at time t_i, then a Boolean value False is assigned to node v_j.
5. line 31 allows to traverse all the child nodes of node v_j to check if they can be recursively traversed by the Algorithm 2.
6. line 32 checks if the child node has not yet been traversed and if all its parent nodes are True at time t_i. In this case, Algorithm 2 is called recursively for this node.
7. line 34 checks if the child node is a literal. If this node has not yet been traversed and at least one of its parent nodes is True, then Algorithm 2 is called recursively for this child node.

Let us take as an example the traversal of the attack graph in the remote working use case at time t_{169}. The result of the simulation is visible on the figure 50. This time slice corresponds to the moment when the user connects its computer to the home network that has already been compromised by the attacker. In this use case, the simulation starts on 2021-01-04 00:00:00 and ends on 2021-04-24 23:59:59 with a step τ of 3600 seconds. The nodes which have been assigned a Boolean value during the initialization phase are progressively traversed with Algorithm 2. The graph traversal works as follows:

1. node v_1 is the first node to be traversed. A Boolean value True was assigned to it at time t_{169} during the initialization phase and this value is therefore retained. Its only child node,

node v_3, cannot yet be traversed because it represents a reasoning rule and all its parents have not yet been traversed.

2. node v_2 is the next node to be traversed. A Boolean value True was assigned to it at time t_{169} during the initialization phase and this value is therefore retained. Its only child node, node v_3, can this time be traversed because all its parents have been traversed and are all True at time t_{169}. Algorithm 2 is therefore called recursively to traverse this node.

3. the node v_3 represents a reasoning rule and all its parent nodes are True at time t_{169}. The clock associated with this node is equal to 0 and the rule can therefore be activated immediately. A Boolean value True is assigned to node v_3. Its only child node, node v_4, has not yet been traversed. This node represents a literal and at least one of its parents is True, so Algorithm 2 can be called recursively.

4. a Boolean value True is assigned to node v_4. Its only child node, node v_7, cannot yet be traversed because it is a reasoning rule and all its parents have not yet been traversed.

5. node v_5 is the next node to be traversed. A Boolean value True was assigned to it at time t_{169} during the initialization phase and this value is therefore retained. Its only child node, node v_7, still cannot be traversed because all its parent nodes have not yet been traversed.

6. node v_6 is the next node to be traversed. A Boolean value True was assigned to it at time t_{169} during the initialization phase and this value is therefore retained. Its only child node, node v_7, can be traversed this time because all its parent nodes have been traversed.

7. Algorithm 2 is thus called recursively to deal with node v_7. The node v_7 represents a reasoning rule whose clock value is associated with a probability distribution. During the initialization phase, this distribution was used to sample a value and assign it to the clock. In this example, the clock is equal to 5,000 seconds. All parent nodes of node v_7 being True, the clock is decremented by the duration of one simulation step and is finally equal to 1,400 seconds. The value of the clock is not less than 0 and a Boolean value False is therefore assigned to node v_7. Therefore its only child node, node v_8, cannot be traversed.

8. Algorithm 2 will continue to be called for each node that has been assigned a Boolean value during the initialization phase, but without activating new reasoning rules.

9. finally, a Boolean value False is assigned to all nodes of the attack graph that have not been traversed at time t_{169}.

In the next section, we show how the matrix M^2 allows to evaluate the risk in a system.

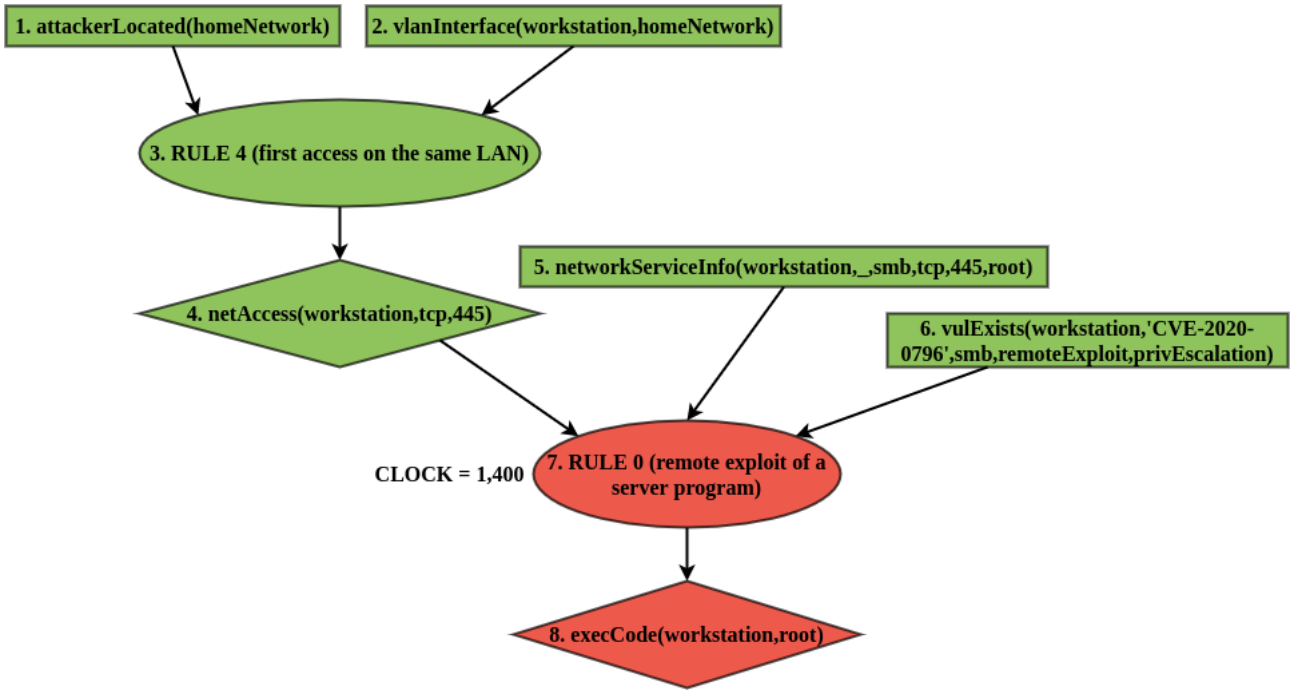


Figure 50: Results of the simulation at time t_{169}

II.3.2 Metric calculation

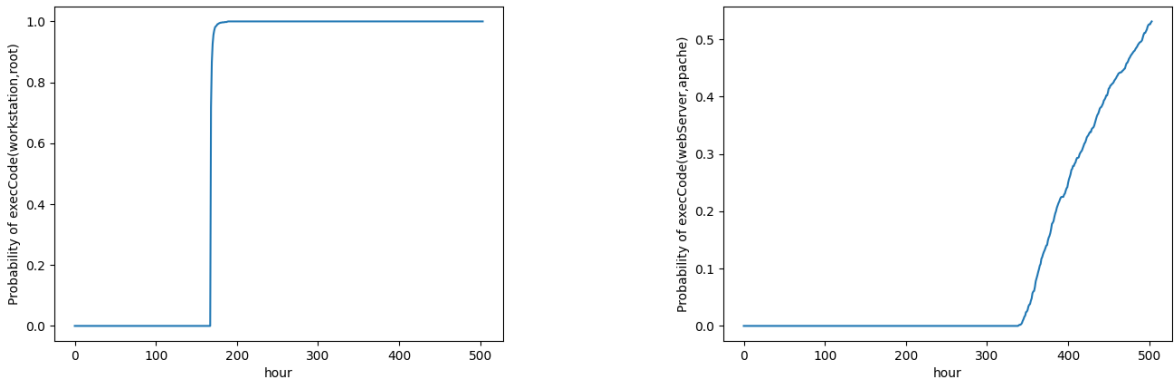
As a reminder, the matrix $M^2 = (a_{ij})$ contains the Boolean values of all the nodes v_j of the attack graph for each time slice t_i. The objective is to approximate the probability for a node v_j to be True during the time slice t_i. Let $X_{v_j,t_i} \in \{True, False\}$ be a random variable representing the Boolean value assigned to a node v_j during the time slice t_i after the execution of a simulation, such as $X_{v_j,t_i} = a_{ij}$. After performing d simulations, we obtain a new tridimensional matrix $M^3 = (a_{ijk}), k \in K$ with $K = \{s \in \mathbb{N} | 1 \leq s \leq d\}$. An element of the matrix a_{ijk} is equal to 1 if the node v_j of the simulation s_k is True during the time slice t_i, and 0 otherwise. Let S_{v_j,t_i} be the set of d random variables $X^k_{v_j,t_i}$ obtained after the realization of all the simulations such that $S_{v_j,t_i} = \{X^1_{v_j,t_i}, \cdots, X^d_{v_j,t_i}\}$, with $X^k_{v_j,t_i} = a_{ijk}$. It is possible to approximate the probability for a node v_j to be True during time slice t_i from the matrix M^3 with the following equation:

$$P(X_{v_j,t_i}) = \overline{S_{v_j,t_i}} = \frac{1}{d} \times |S^T_{v_j,t_i}| \text{ ,with } S^T_{v_j,t_i} = \{X^k_{v_j,t_i} | X^k_{v_j,t_i} = True\} \tag{55}$$

A new matrix $R^2 = (r_{ij})$ can then be defined such that $r_{ij} = P(X_{v_j,t_i})$. This metric allows to graphically represent the evolution of the probability that each node of the attack graph is

true.

In the remote working use case, we chose to perform 1,000 simulations to get an accurate result. Figure 51a represents the evolution of the compromise probability of the user station. As illustrated, the attacker does not manage to compromise the user workstation the first week because he does not have access to the company's internal network. A rapid increase in probability is visible at the beginning of the second week when the user station is connected to the home network. At this point, the attacker who has already compromised this network, accesses the vulnerable SMB service on the computer and easily exploits the vulnerability. The exploitation of this vulnerability is greatly simplified thanks to the release of an exploit code, which allows the attacker to execute malicious code on the user's machine. This code will continue to run until the end of the simulation because the attack has not been detected by the security teams. Figure 51b illustrates the evolution of the compromise probability of the web server. As we can see, the probability was zero during the first two weeks. Indeed, the attacker only succeed to compromise the user workstation at the beginning of the second week and has to wait until the third week for the machine to be reconnected to the company's internal network. At this moment, the attacker has access to the web service from the user workstation over which he has taken remote control. The vulnerability present on this service is difficult to exploit, so the attacker has only an 50% probability of having compromised the web server by the end of the third week.



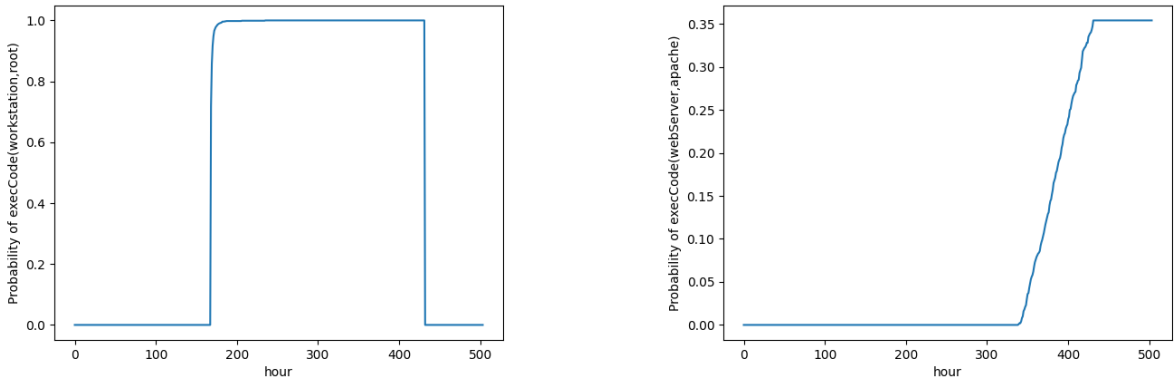
(a) Evolution of the workstation compromise probability

(b) Evolution of the web server compromise probability

Figure 51: Evolution of the compromise probability of the system components when no patch is applied

We performed other simulations from the remote working use case by changing some parameters. We modeled the workstation wipe between 2021-01-02 00:00:00 and 2021-01-22 23:59:59, after the security teams has detected the attack. Figure 52a represents the evolution of the compromise probability of the user station. We can see that the probability of compromise of

the user station drops to zero on 2021-01-02 00:00:00 and does not increase anymore because the attacker no longer has access to the vulnerable SMB service. Figure 51b illustrates the evolution of the compromise probability of the web server. We can see that the workstation wipe reduces the probability of web server compromise to 35%.

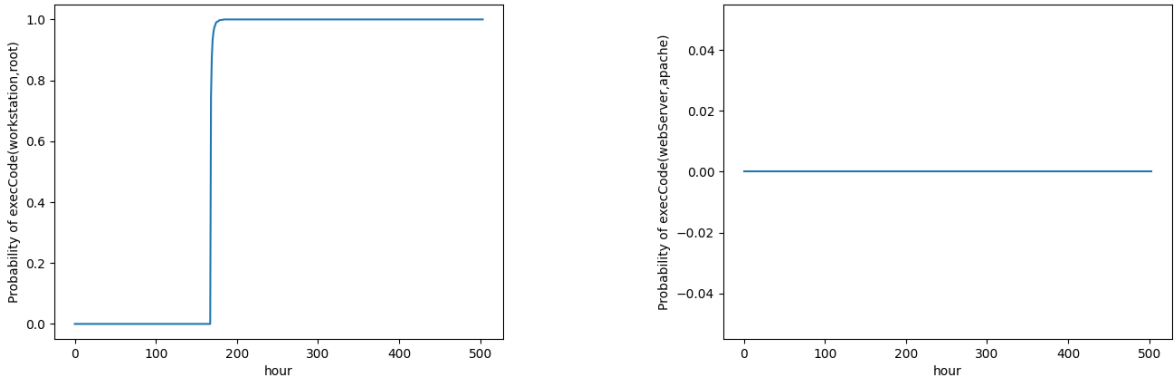


(a) Evolution of the workstation compromise probability

(b) Evolution of the web server compromise probability

Figure 52: Evolution of the compromise probability of the system components when the user station is wiped in the third week

We modeled the patch of the vulnerability present on the web service. Figure 53a represents the evolution of the compromise probability of the user station. We see that the application of the patch on the web server has no influence on this probability. Figure 53b illustrates the evolution of the compromise probability of the web server. We can see that the attacker is no longer able to compromise the web server. When the vulnerability present on the Remote Desktop Protocol (RDP) service of the user station is patched, the attacker can no longer compromise anything because he cannot use the user station to access the company's internal network.

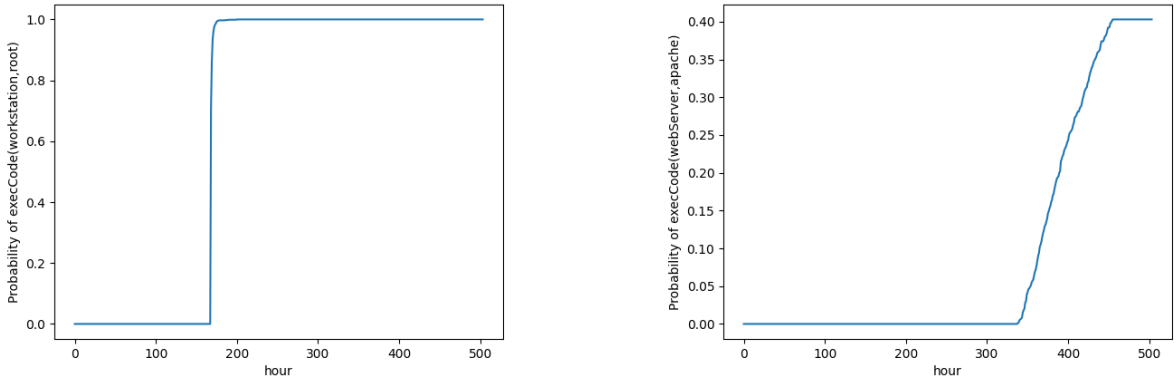


(a) Evolution of the workstation compromise probability

(b) Evolution of the web server compromise probability

Figure 53: Evolution of the compromise probability of the system components when the vulnerability present on the web service is patched

On the other hand, we performed simulations where the vulnerabilities are patched during the time period studied. We modeled the patch of the vulnerability present on the web service on 2021-01-23 00:00:00. Figure 54a represents the evolution of the compromise probability of the user station. We note that the patch has no impact on this probability. Figure 54b illustrates the evolution of the compromise probability of the web server. The probability has stopped increasing on 2021-01-23 00:00:00 and stabilized at 40%.



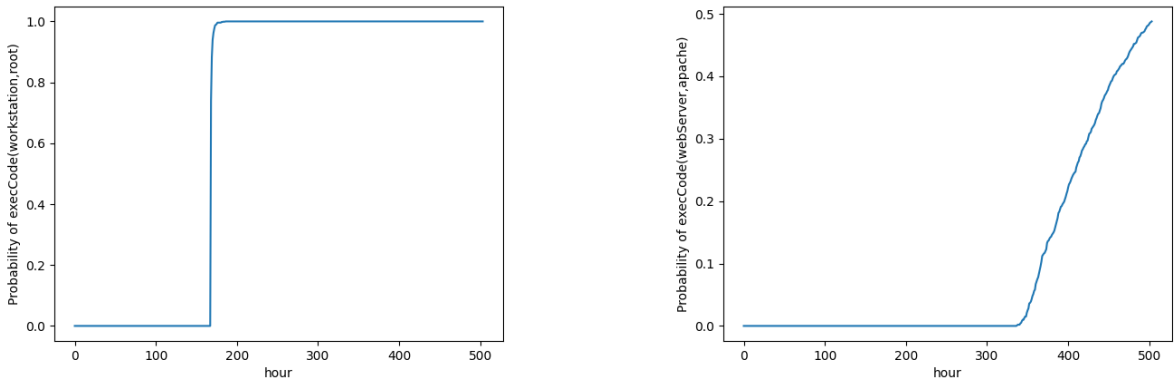
(a) Evolution of the workstation compromise probability

(b) Evolution of the web server compromise probability

Figure 54: Evolution of the compromise probability of the system components when the vulnerability on the web service is patched on 2021-01-23 00:00:00

We modeled the patch of the vulnerability present on the SMB service of the workstation on 2021-01-13 00:00:00. Figure 55a represents the evolution of the compromise probability of the user station. We note that the patch has no impact on this probability because the vulnerability

is patched too late while it is very easily exploitable. Therefore, there is also no difference in the probability of web server compromise, as we can see in Figure 55b.



(a) Evolution of the workstation compromise probability

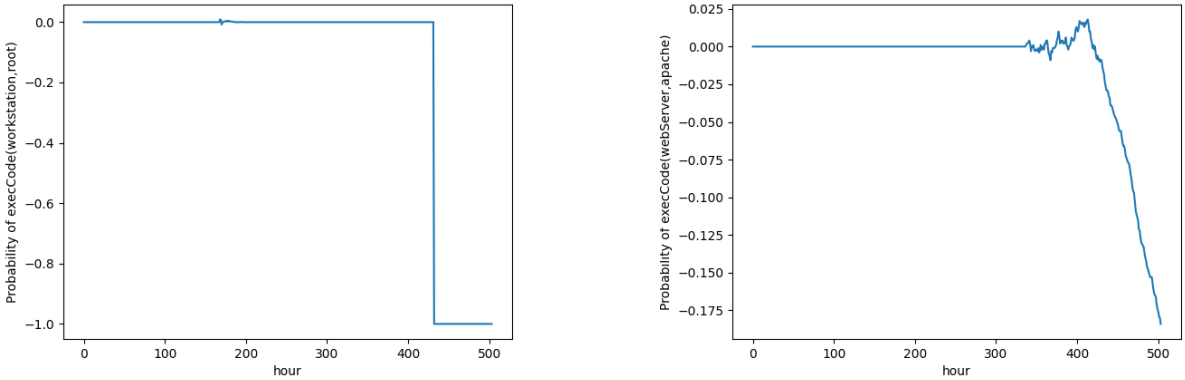
(b) Evolution of the web server compromise probability

Figure 55: Evolution of the compromise probability of the system components when the SMB vulnerability is patched on 2021-01-13 00:00:00

A second metric, noted TTC (Time To Compromise), has been defined to measure the time required for a node of the attack graph to be True with a probability higher than a threshold value p_s defined by the user. This time is calculated with equation (56) from the matrix R^2 containing the results of the simulation. TTC values can be represented with a color gradient to display a heatmap of the system showing the components most impacted by the attack.

$$TTC(v_j) = \frac{1}{i_s \times \tau} \text{ , such as } \nexists r_{ij} \geq p_s \text{ , with } i < i_s \text{ and } r_{i_s+1\,j} > p_s \tag{56}$$

A third metric allows to measure the impact of a configuration change in the system by comparing the probabilities of compromise of the system components between two simulations. For example, Figure 56a and 56b show the comparison of the probability of compromise of the user workstation and the web server, between the simulation without patching and the one with the wipe of the workstation on 2021-01-02 00:00:00. In Figure 56a, we can see that there is no difference in the probability of compromise the workstation before the station is wiped. After the wipe of the workstation, the difference is 100% until the end of the third week. In Figure 56b, we can also see that there is no difference in the probability of compromise the web server before the station is wiped. After the wipe of the workstation, the difference gradually increases until the end of the third week, reaching 18%. This metric can also be used to measure the impact of one or a set of vulnerabilities. To do this, two simulations are compared, one with the vulnerabilities and the other with the patches.



(a) Evolution of the workstation compromise probability

(b) Evolution of the web server compromise probability

Figure 56: Evolution of the difference in probability between the simulation without patch and the simulation with the user station wipe

In the next section, we will see the performance of our solution by evaluating the complexity of the algorithms used and by performing several benchmarks.

Chapter III

Performance evaluation of the solution

III.1 Evaluation of the algorithms complexity

III.1.1 Definition of variables

The proposed solution should be able to analyze the security of large systems composed of several thousands of components. It is therefore essential that the worst-case time complexity of the algorithms used is adapted to this context. But first, let's look at the difficulty of modeling the system. Indeed, one of the first difficulties we encounter when we are interested in the analysis of the security of complex systems is to be able to model them. In our solution, the modeling step consists in writing a set of temporal literals and reasoning rules. This step can quickly become difficult when thousands of literals must be written to represent the properties of all the components of the system. Therefore, we have set up a system of variables that allows us to simplify the writing of these literals.

A variable represents a type of object in the system and can take several values. In our model, we have defined the variable *VAR_ADDC* which represents the set of domain controllers present in the system such that *VAR_ADDC* = [*'ADDC-PA'*, *'ADDC-ST'*, *'ADDC-MA'*, *'ADDC-LY'*, *'ADDC-BR'*]. A variable must start with an uppercase letter to differentiate it from a constant. These variables can then be used in the parameters of a literal. For example, the literal *vulExists*(*VAR-ADDC*, *'CVE-2022-26809'*, *rpc*, *remoteExploit*, *privEscalation*) indicates that the vulnerability *CVE-2022-26809* is present on the Remote Procedure Call (RPC) service of all domain controllers in the system. Several variables can be used in the same literal. In this case, a Cartesian product is realized between the sets of values represented by the variables. For example the literal *vulExists*(*VAR-ADDC*, *VAR-VULN-RPC*, *rpc*, *remoteExploit*, *privEscalation*), with *VAR-VULN-RPC* = [*'CVE-2022-26809'*, *'CVE-2021-27091'*] indicates that both vulnerabilities *CVE-2022-26809* and *CVE-2021-27091* are present on the RPC service of all domain controllers in the system. The literals containing variables are then replaced by literals containing the value of the variables in parameter, before being used by the MulVAL framework.

III.1.2 Analysis of the complexity of the attack graph generation algorithm

The complexity of the MulVAL attack graph generation algorithm has already been studied in [34]. The complexity of the evaluation of a Datalog program in XSB depends essentially on the evaluation of the reasoning rules. This evaluation is performed in two steps. Let's take the

following rule as an example.

netAccess(Attacker, H2, Protocol, Port) :-
execCode(Attacker, H1, User),
hacl(H1, H2, Protocol, Port)

In this example, XSB first lists all the machines on which the attacker can execute code, then lists all the accesses allowed between two machines on the network with the port and protocol specified in the rule. Once this step is done, XSB will look for the literals that match each other and that allow to infer the deduced literal *netAccess*. This step is however well managed in XSB with the use of hash tables and can therefore be ignored in the evaluation of the complexity. Therefore, the evaluation of the complexity of a Datalog program in XSB can be simplified to the first step which consists in computing all the literals that correspond to the parameters of the reasoning rule.

In the default rules defined in the MulVAL framework, it is the reasoning rule presented above that corresponds to the worst case. Indeed, to check the literals corresponding to the parameter *hacl(H1, H2, Protocol, Port)*, it is necessary to list all the possible connections between two machines of the system, which makes in the worst case $(n-1)^2$ connections, with n the number of machines present on the network. We were careful when creating our reasoning rules to avoid increasing the complexity of the Datalog program by not creating a rule with more than two different machines in a rule's parameter literal.

If we add the complexity of the attack graph generation algorithm, the overall worst-case complexity of MulVAL algorithm is $O(N^2 \times log(N))$, with N the number of hosts, according to [34]. Once the attack graph is generated with MulVAl, dynamic properties such as clocks and time intervals are added to each node. In our implementation, the graph is traversed a first time to be loaded in memory and a second time to add the dynamic properties. The worst-case time complexity of this step is therefore $O(2 \times n)$, with n the size of the attack graph.

III.1.3 Optimization of the attack graph size

The size of the attack graph generated by MulVAL evolves in the worst case quadratically with respect to the number of machines in the system. The size of the attack graph has a significant impact on the performance of our solution when our attack simulation algorithm must traverse it. We have therefore aimed to control the size of the attack graph generated by

MulVAL.

A large number of literals in an attack graph are not analyzed by security teams and are only used as intermediate nodes. For example, the predicates execCode, availability, confidentiality and ddos are used to measure the success of the attacker in compromising the system. But the predicates netAccess and lanAccess have no interest for security teams and are only used to describe intermediate steps in the construction of the attack graph.

We therefore looked for a way to reduce the size of the attack graph by removing some intermediate nodes. When an intermediate node v_i is deleted, the reconstruction of the attack graph consists of deleting the parent node and the child nodes of v_i, and then creating a new reasoning rule for each path between a grandparent node of v_i with the grandchild node of v_i.

To explain our approach, we will take as an example the attack graph visible in Figure 57. The first reasoning rule, represented by the node v_5, allows to deduce that the attacker has access to a network service. The second reasoning rule, represented by the node v_9, allows to deduce that the attacker can execute code on the machine. In this example, the information of interest to the security teams is the possible compromise of the system by the attacker, represented by the literal *execCode* and the node v_6 in the graph. The literal *netAccess* is therefore only an intermediate node, and it is represented by the node v_{10} in the graph.

We propose a solution that reduces the size of the attack graph by removing the intermediate literal v_6, which gives the result visible in Figure 58. The size of the graph has been reduced from 10 to 8 nodes. However, in some cases, deleting an intermediate node can actually increase the size of the graph. Figure 59 shows three cases where an intermediate node is removed. In the first case, the deletion of node v_2 reduced the size of the attack graph from 5 to 3 nodes. In the second case, deleting the intermediate node v_3 does not change the size of the attack graph. Finally, in the third case, the deletion of node v_4 increases the size of the attack graph from 7 to 9 nodes.

To know if deleting an intermediate node will reduce the size of the attack graph, we need to compare the number of nodes deleted and created during the reconstruction of the graph. Let v_i be an intermediate node that we want to delete. So initially we have node v_i, its parents $indegree(v_i)$ and its children $outdegree(v_i)$, which makes a total of $indegree(v_i) + outdegree(v_i) + 1$ nodes. After replacing the intermediate node v_i, we have $indegree(v_i) \times outdegree(v_i)$ new nodes that are created. Therefore, to find out whether deleting an intermediate node v_i will reduce the size of the attack graph, the condition (1) must be verified. We have developed an algorithm to traverse an attack graph generated by MulVAl to remove intermediate nodes when the condition (1) is verified.

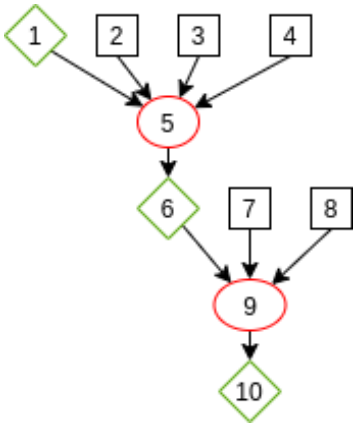


Figure 57: Part of an attack graph

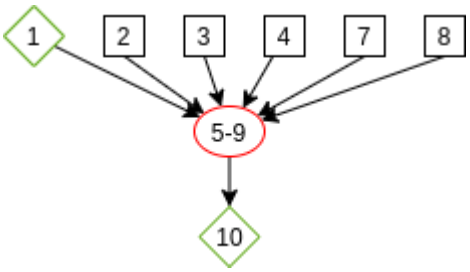


Figure 58: Part of an attack graph after removing the node v_6

Condition 1 $indegree(v_i) \times outdegree(v_i) < indegree(v_i) + outdegree(v_i) + 1$

In some cases, the addition of an intermediate node reduces the size of the attack graph. For example, we faced a problem of rapid growth in the size of the graph when we started to model the possible connections between the machines present on the same network with the following reasoning rule:

$$\begin{gathered} netAccess(HostB,\ \_,\ \_)\ :\text{-} \\ vlanInterface(HostA,\ Vlan), \\ vlanInterface(HostB,\ Vlan), \\ execCode(HostA,\ Perm) \end{gathered}$$

This reasoning rule had the effect of creating a new literal *netAccess(Host, Prot, Port)* for each pair of hosts present on the same network. But there are $(n-1)^2$ possible connections between n machines connected on the same network, which made the size of the attack graph increase quadratically. To deal with this problem, we have added a new intermediate node *lanAccess(vlan)* that can be deduced with the following reasoning rule:

$$\begin{gathered} lanAccess(Vlan)\ :\text{-} \\ execCode(Host,\ Perm), \\ vlanInterface(Host,\ Vlan) \end{gathered}$$

Then we modified the reasoning rule to derive the literal *netAccess(Host, Prot, Port)* such

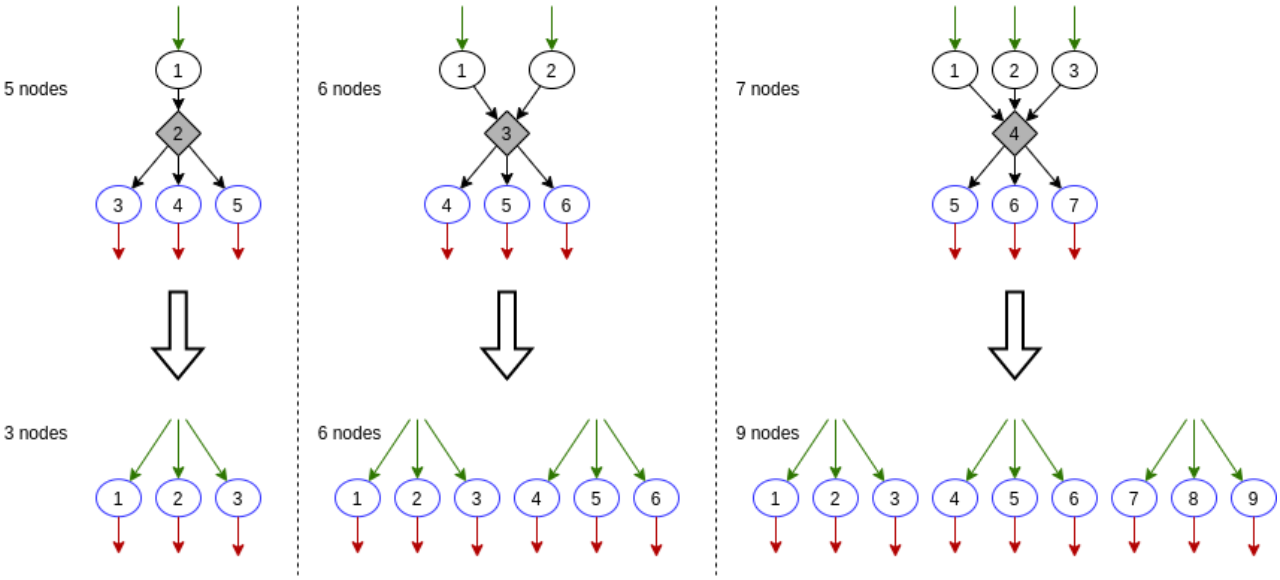


Figure 59: Three cases of deletion of an intermediate node in an attack graph

that:

$$netAccess(Host,\ Prot,\ Port)\ :\text{-}$$
$$lanAccess(Vlan),$$
$$vlanInterface(Host,\ Vlan)$$

Figure 60 illustrates how adding this intermediate node reduces the size of the attack graph.

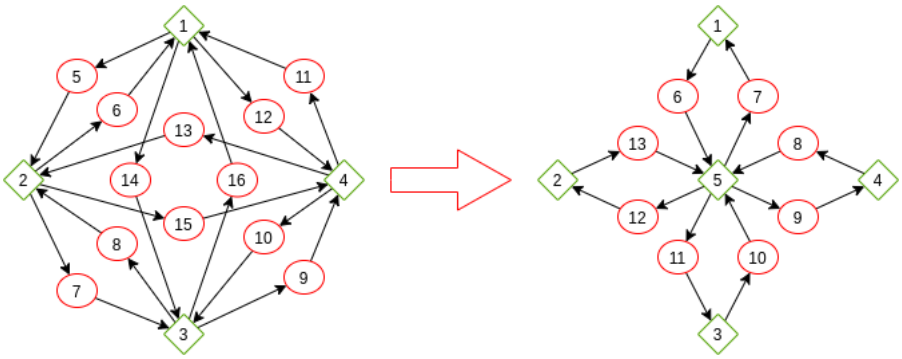


Figure 60: Addition of the intermediate node v_5 *lanAccess* to reduce the size of the attack graph

III.1.4 Analysis of the complexity of the simulation algorithm

We also studied the complexity of our graph traversal Algorithm 2 used in the simulation. Let $G = (V,E)$ be an attack graph, with $V = L_i \cup L_d \cup R$ and $E = E_r \cup E_l$ such that L_i is the set of initial literals, L_d the set of inferred literals, R the set of reasoning rules, E_r the incoming arcs to a reasoning rule and E_l the incoming arcs to a literal. The Algorithm 2 is called for all nodes v_i which have been assigned a Boolean value during the initialization phase. The number of these nodes is equal in the worst case to the number of literals present in the graph, that is $|L_i \cup L_d|$. In the worst case, the attack graph to be traversed is constructed such that an arc $e = (v_j, v'_j) \in E_r$ exists for all nodes $v_j \in (L_i \cup L_d)$ and $v'_j \in R$. We will first look at the number of checks made on line 31 of Algorithm 2 to know if a child node can be traversed. For a node $v_j \in (L_i \cup L_d)$, all its child nodes $v'_j \in R$ are verified, which makes $|R| \times |(L_i \cup L_d)|$ checks. For a node $v_j \in R$, it is its single child node $v'_j \in (L_i \cup L_d)$ that is verified, which makes $|R|$ checks. Concerning node traversal by Algorithm 2, a node $v_j \in R$ can only be traversed once during a recursive call. A node $v_j \in (L_i \cup L_d)$ can be traversed once recursively by Algorithm 2 and another time when traversing the nodes having a Boolean value assigned during the initialization phase. We thus have a total of $2 \times |(L_i \cup L_d)| + |R|$ nodes traversed by Algorithm 2. As the attack graph is traversed for each time slice $t_i | i \in I$, we thus have a worst-case time complexity of Algorithm 2 equal to $O([(2 + |R|) \times |(L_i \cup L_d)| + 2 \times |R|] \times |I|)$. The different simulations that need to be performed can be executed in parallel on several CPU cores or on several machines, which does not increase the overall algorithmic complexity. If we assume that the attack graph contains as many literals as reasoning rules, we can simplify the complexity to $O([\frac{n^2}{2} + n] \times |I|)$, with n the size of the graph.

III.2 Results of benchmarks

III.2.1 Test environment

We also studied the real performance of our algorithms by performing a series of benchmarks in different configurations. We used the network presented in the use case of remote working where one third of the users work from home every week. We measured the performance of our solution in 6 different configurations:

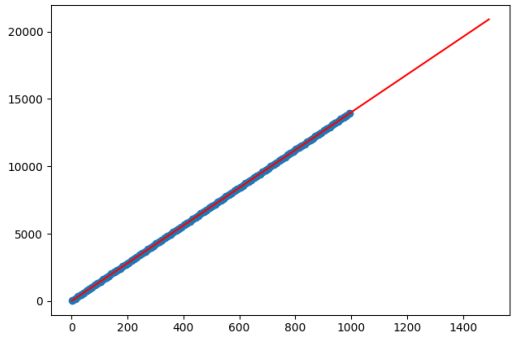
- with a variation in the number of hosts and without optimization;
- with a variation in the number of vulnerabilities and without optimization;

- with a variation in the number of initial literals and without optimization;
- with a variation in the number of hosts and with optimization;
- with a variation in the number of vulnerabilities and with optimization;
- with a variation in the number of initial literals and with optimization;

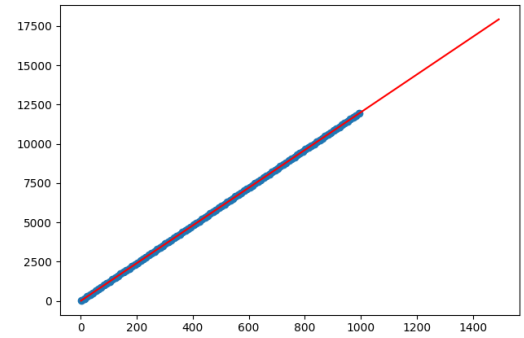
We measured for each execution of our solution the number of hosts or vulnerabilities in the system, the number of initial literals, the size of the attack graph, the computation time of the optimization if it is performed, the execution time of the MulVAL algorithm and the execution time of the simulation. The tests were performed on an Ubuntu 20.04 virtual machine with a Linux kernel 5.15.0-41-generic, and with 16 CPUs at 2 GHz and 64 GB of memory. A least squares polynomial regression is used on the benchmark results to assess the degree of the polynomial that best fits the observed data, and the coefficient of determination R-squared is used to measure the quality of the regression.

III.2.2 Evolution of the attack graph size

We measured the evolution of the size of the attack graph in several configurations. Figure 61a shows how the size changes as the number of hosts increases. By performing a polynomial regression, we obtained that the polynomial $14x+12$ perfectly matches the observed data with a R-squared of 1. Figure 61b shows the evolution of the attack graph size with optimization. The polynomial regression indicates that the polynomial $12x+8$ perfectly matches the observed data with a R-squared of 1. We can clearly see an improvement in the size of the graph thanks to our optimization algorithm. This improvement is more important as the number of hosts increases, with a difference in size equal to $2n+4$, with n the number of machines in the system.



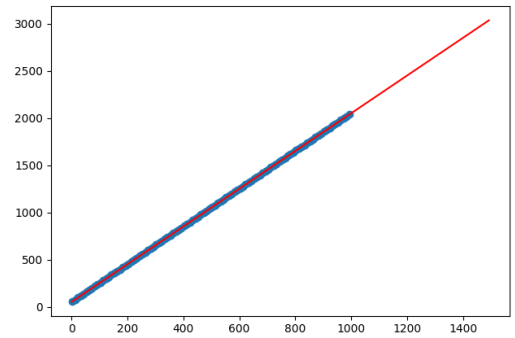
(a) Evolution of the attack graph size without optimization



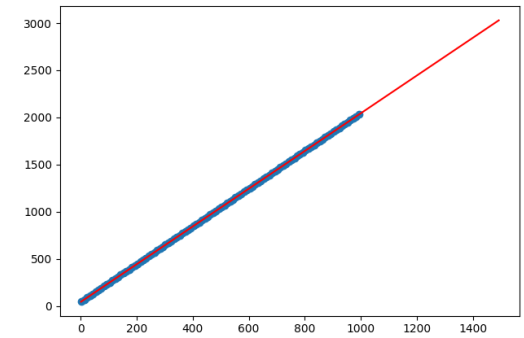
(b) Evolution of the attack graph size with optimization

Figure 61: Benchmark results in blue and polynomial regression in red showing the evolution of the attack graph size in relation to the number of hosts present in the system

We made the same measures but this time according to the number of vulnerabilities present on the web service. Figure 62a shows this evolution when the graph size is not optimized. The polynomial regression indicates that the polynomial $2x+52$ perfectly matches the observed data with a R-squared of 1. Figure 62b shows this evolution when the size of the attack graph is optimized. The polynomial regression indicates that the polynomial $2x+44$ perfectly matches the observed data with a R-squared of 1. We see here that the improvement in the size of the attack graph brought by our optimization algorithm is much less important. It is 8 nodes and does not evolve with the number of vulnerabilities.



(a) Evolution of the attack graph size without optimization

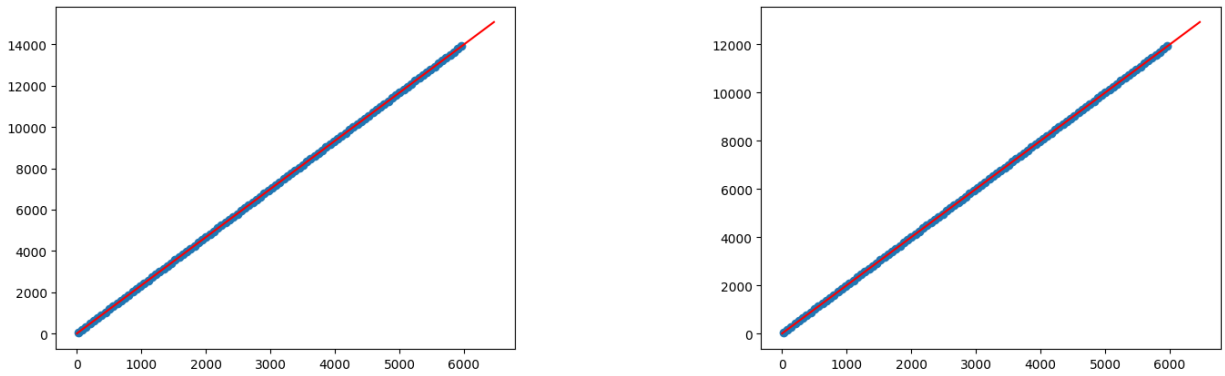


(b) Evolution of the attack graph size with optimization

Figure 62: Benchmark results in blue and polynomial regression in red showing the evolution of the attack graph size in relation to the number of vulnerabilities present in the system

Finally, the same measures were performed as a function of the numbers of literals given as parameters to MulVAL. Figure 63a shows the results of this evolution without any optimization.

The polynomial regression that was performed on the data issued from the benchmark shows that the size of the attack graph evolves linearly as a function of the number of input literals, following the polynomial $2.33 \times x - 2$, with a R-squared of 1. Figure 63b shows the benchmark results of the evolution of the attack graph size as a function of the number of literals and after applying the optimization algorithm. We can see that the size also evolves linearly by following the polynomial $2 \times x - 4$, with a R-squared of 1. This evolution is slower thanks to the optimization performed on the graph, with a difference that evolves linearly following the polynomial $0.33 \times x + 2$.



(a) Evolution of the attack graph size without optimization

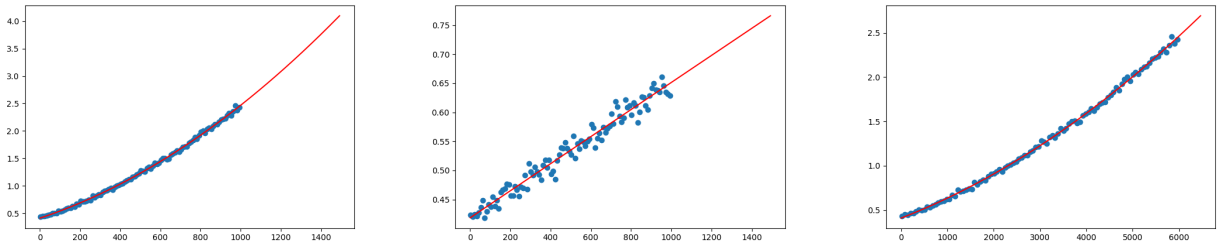
(b) Evolution of the attack graph size with optimization

Figure 63: Benchmark results in blue and polynomial regression in red showing the evolution of the attack graph size in relation to the number of initial literals

III.2.3 Evolution of the execution time of the attack graph generation algorithm

We will now see how the execution time of the MulVAL algorithm that generates the attack graph evolves. Figure 64a shows the evolution of the execution time as a function of the number of hosts present on the network. The polynomial regression indicates that the polynomial $8.39 \times 10^{-07}x^2 + 1.22 \times 10^{-03}x + 4.11 \times 10^{-01}$ fits the observed data well with a R-squared of 0.99. Figure 64b shows this same evolution but this time as a function of the number of vulnerabilities present on the web service. The polynomial regression indicates that the polynomial $2.33 \times 10^{-04}x + 4.18 \times 10^{-01}$ corresponds to the observed data with a R-squared of 0.97. It can be seen that the number of vulnerabilities present in the system has less impact on MulVAL's computation time than the number of hosts. Finally, we have measured the evolution of the execution time of the MulVAL algorithm as a function of the number of initial

literals. The benchmark results can be seen on Figure 64c and show that the execution time evolves quadratically with respect to the number of initial literals following the polynomial $2.33 \times 10^{-08}x^2 + 2.02 \times 10^{-04}x + 0.41$, with an R-squared of 0.99.



(a) In relation to the number of hosts present in the system

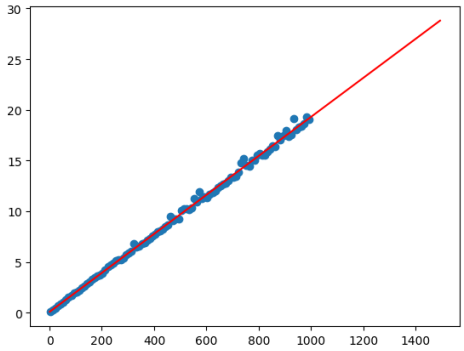
(b) In relation to the number of vulnerabilities present in the system

(c) In relation to the number of initial literals

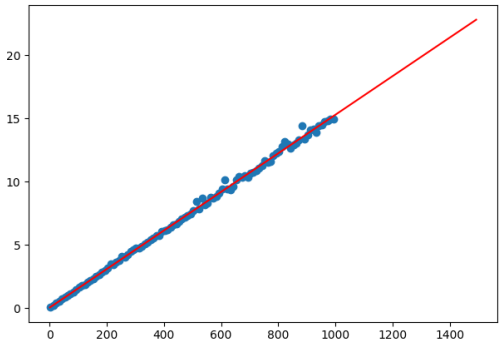
Figure 64: Benchmark results in blue and polynomial regression in red showing the evolution of the execution time of the MulVAL algorithm

III.2.4 Evolution of the execution time of the simulation algorithm

Finally, we were interested in measuring the evolution of the execution time of the simulation algorithm in different configurations. Figure 65a shows the evolution of the computation time as a function of the number of hosts present on the network. The polynomial regression indicates that the polynomial $0.019x + 0.068$ fits the observed data well with a R-squared of 0.99. Figure 65b shows the same evolution but in the case where the size of the attack graph is reduced thanks to the optimization algorithm. The polynomial regression indicates that the polynomial $0.015x + 0.027$ matches the observed data with a R-squared of 0.99. However, the time required to optimize the graph must be considered if we want to compare these results. Figure 66 shows the evolution of the execution time of the optimization algorithm as a function of the number of hosts present in the system. The polynomial regression indicates that the polynomial $2.13 \times 10^{-06}x^2 + 4.5 \times 10^{-04}x - 3.38 \times 10^{-02}$ matches the observed data with a R-squared of 0.99. The total execution time including the graph optimization and the simulation is therefore equal to $2.13 \times 10^{-06}x^2 + 1.55 \times 10^{-02}x - 6.8 \times 10^{-03}$. If we compare this equation with the one representing the execution time of the simulation without optimization, we notice that for all $x > 1687$, the optimization is not interesting anymore. It would be interesting to verify in future works if the optimization algorithm cannot be improved to be used on very large systems. In the next section, we will test our model on a complex use case representing an IT system of a huge company.



(a) Evolution of the execution time of the simulation algorithm without optimization



(b) Evolution of the execution time of the simulation algorithm with optimization

Figure 65: Benchmark results in blue and polynomial regression in red showing the evolution of the execution time of the simulation algorithm in relation to the number of hosts present in the system

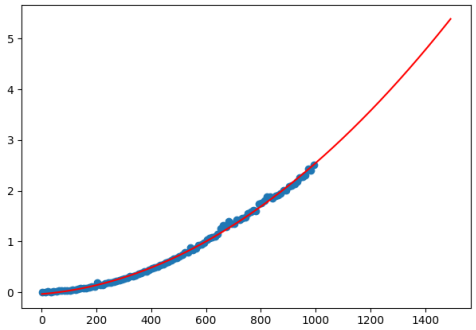


Figure 66: Evolution of the execution time of the attack graph size optimization algorithm in relation to the number of hosts present in the system

III.3 Results of the scalability test

III.3.1 Test environment

We tested our model on a complex use case to validate its ability to analyze the security of large systems. Our use case represents the IT system of a large French company present on several sites throughout the country. The network is distributed across the cities of Paris, Brest, Strasbourg, Lyon and Marseille. These different networks are connected through an inter-site Virtual Private Network (VPN). The network is segmented into several Virtual Local Area Networks (VLANs):

- VLAN 100 is used to connect the administrators' machines and to access the administration interfaces of the different components of the system.
- VLAN 200 is a classified network that does not have access to the Internet or to internal company services.
- VLAN 300 is the unclassified user network.
- VLAN 400 is the company's service network where all the organization's servers are connected.
- VLAN 500 is the Demilitarized Zone (DMZ) network. It is on this network that the servers accessible from the outside are connected.
- VLAN 600 is a guest network dedicated to people outside the company who wish to access the Internet. This network is accessible through a Wireless Access Point (WAP) present in each lobby of the company's sites.
- VLAN 700 is the company's remote access network that is accessible through a VPN. This network allows users connected remotely from home to have access to the same services as if they were connected on the internal unclassified user network.
- VLAN 800 is the VPN network that interconnects all the sites of the organization.

The network configuration is identical for each site. The internal firewall allows to interconnect the different VLANs and to manage the access rights to the different services of the company. These servers are all connected to the service network (VLAN 400) of the Paris site. The network configuration is described as fllowing.

- The internal firewall allows to interconnect the different VLANs and to manage the access rights to the different services of the company.
- Internet access from the unclassified user network is done through a Blue Coat proxy. The access to the various services of the company is always done through a proxy F5 BIG-IP. This proxy allows to perform load balancing and to distribute the incoming connections on the different instances of the backend servers.
- A first server is used for Human Resources (HR) management. It is an Apache Tomcat service hosted on a Red Hat Enterprise 7. There are three instances of this server that all have access to the same Oracle MySQL database server.

- There are 4 Red Hat Enterprise 7 application servers used by users to manage the company's business processes. Two of these servers host an Apache Tomcat web service distributed over 3 instances. The first one has access to the MySQL Oracle and PostgreSQL database servers, while the second one only has access to the PostgreSQL server. The third application server is distributed over 2 instances, each hosting an Apache Axis web service. Both instances have access to an Oracle database. The fourth application server is distributed over two instances, each hosting a Splunk service. This service allows to manage the logs coming from the different components of the system. Each instance has access to the MySQL Oracle database. The externally accessible Red Hat Enterprise 7 web server is connected to the DMZ network of the Paris site.
- An F5 BIG-IP proxy is used to distribute the load over three web server instances, each hosting an Apache Tomcat service. Each of these instances has access to the MySQL Oracle database server.
- An Exchange 2019 server is used to send and receive mail from the connected stations on the unclassified user network.
- Each user has a Cisco Internet Protocol (IP) phone. The phones are interconnected through Autocom servers located on each site of the organization.
- A Windows 2016 Domain Controller (DC) is present on each site and is used to manage the access rights of the different users. Each DC is synchronized with the one located in Paris in order to replicate the data of all the sites of the organization. There is a forest *.admin.corpo.int* for administrators and a forest *.user.corpo.int* for users.
- People outside the company connected to the guest network have access to the Internet, which is directly routed through the company's internal and external firewalls without going through the Blue Coat proxy. Users working remotely must connect to the Stormshield VPN on the external firewall to access the various company services. The interconnection of the different sites goes through a dedicated physical network and is protected by a VPN tunnel established between the external firewalls.
- HP A3 multifunction printers are present on each site of the company and can be used by all users.
- The user stations can be either laptops or desktop computers. The Windows operating system is installed on all these stations but with different versions. There are Windows XP 2002 SP3, Windows 7 7601 SP1 and Windows 10 1803. Only laptops can be used by users who work remotely. The interconnection of the different networks is managed by Cisco switches.

- There are 30,000 employees in this company, each with a computer and a phone. 1% of these employees are network administrators, 19% work on a classified network and the remaining 80% are connected to the unclassified network. Among users who do not work on classified data, 10% of them work remotely 2 days a week. Table 17 shows a complete inventory of the equipment present on the company's network with their version numbers.
- Several vulnerabilities are present on the system components. They are all listed in Table 16 with their main characteristics.

III.3.2 Presentation of the results

We performed our tests in 6 different configurations:

- with 6,000 machines connected to the network and 25% of unclassified users working at home;
- with 6,000 machines connected to the network and 10% of unclassified users working at home;
- with 6,000 machines connected to the network and no users working from home;
- with 60,000 machines connected to the network and 25% of unclassified users working at home;
- with 60,000 machines connected to the network and 10% of unclassified users working at home;
- with 60,000 machines connected to the network and no users working from home.

The simulation results for each of these 6 configurations are summarized in Table 13.

We will describe in more detail the results obtained in the configuration with 60,000 machines connected to the corporate network and with 10% of the unclassified users working at home. To model this system, 35,445 literals were written and 175 variables were defined. After the literals containing variables as parameters were expanded, we obtained a total of 206,102 initial literals. We can see here the importance of variables to make the system modeling step easier. Here, they have allowed to divide by 6 the number of initial literals to write. We simulated an attack between 2021-01-04 00:00:00 and 2021-01-10 23:59:59 with a simulation step of 3,600 seconds. 1,000 simulations were executed which allowed us to obtain a very accurate result. We used the same server as for the previous tests, which is an Ubuntu 20.04 virtual machine

Number of assets	6,000			60,000		
Percentage of unclassified users working at home	0%	10%	25%	0%	10%	25%
Number of initial literals	21,728	22,112	22,799	201,728	206,102	212,975
Attack graph generation time (s)	350	532	935	31,147	256,874	617,557
Memory usage during the construction of the attack graph (GB)	1.75	1.78	1.56	2.24	2.15	2.17
Attack graph size (number of nodes)	253,653	304,109	273,506	9,435,135	10,119,121	10,002,158
Time to load the attack graph and to associate the temporal characteristics for each node (s)	11	12	12	314	334	331
Execution time of a simulation (s)	392	610	480	13,500	14,220	14,400
Memory usage during simulation (GB)	2.14	2.29	1.99	18.9	20	19.7
Size of a file containing the result of a simulation (GB)	85	102	92	3,170	3,400	3,361

Table 13: Results of the tests performed on the complex use case with different configurations

with a linux kernel 5.15.0-41-generic, 16 CPUs at 2GHz and 64 GB of memory. The MulVAL algorithm was used to generate an attack graph of 10 millions nodes. The execution took 71 hours and consumed 2.15 GB of memory. The attack graph was then loaded in memory and the dynamic properties of the system were added to each node. This step lasted 334 seconds. The simulations were run in parallel by groups of 16. Each simulation required 237 minutes of computing time. Therefore, the execution of the 1,000 simulations took 247 hours. During the simulation phase, 20 GB of memory was consumed by all the processes. During the execution of a simulation, each process also writes the result directly to a file on the hard disk. We consumed a total of 3.4 TB of storage space.

We can see that the number of initial literals evolves linearly with the number of assets present in the system. This number is multiplied by 9.28, for example, when comparing results without remote working and when the number of assets is multiplied by 10. The increase in the number of remote working users slightly increases the number of initial literals. In the system composed of 6,000 elements, it increases by 1.8% between 0% and 10% of remote working users, and by 3.1% between 10% and 25%.

There is a rapid increase in computation time between the situation with and without remote working users, especially when it concerns the system with 60,000 components. This increase is due to the non-linear complexity of the execution time of the attack graph generation algorithm as a function of the number of initial literals, as shown by the results of the benchmarks previously performed.

The amount of memory used by the attack graph generation algorithm changes slowly as the system size increases. It is, for example, multiplied by 1.28 when the system size is multiplied by 10 and without remote working users.

The size of the generated attack graph increases slightly faster than the number of components in the system. For example, when the system size is multiplied by 10 and without remote working users, the size of the attack graph is multiplied by 37.

The time required to load the attack graph into memory with the addition of the dynamic characteristics increases slightly faster than the system size. For example, it is multiplied by 29 when the size of the system is multiplied by 10 and without remote working users.

The execution time of a simulation increases faster than the system size but at the same rate as the size of the attack graph. For example, when the number of system components is multiplied by 10 and without remote working users, the execution time of a simulation is multiplied by 34 while the size of the attack graph is multiplied by 37.

The memory consumption during the execution of a simulation increases at the same rate as the system size. It is, for example, multiplied by 8.8 when the system size is multiplied by 10 and without remote working users.

Finally, the size of a file containing the result of a simulation increases faster than the system but at the same speed as the size of the attack graph. For example, when the system size is multiplied by 10 and without remote working users, the file size as well as the attack graph size are multiplied by 37.

III.3.3 Conclusion

These results show that our solution is able to assess the security of complex systems in a reasonable time. If all simulations are run in parallel, either on several CPU cores or on different machines, the total execution time is about 12 hours for the configuration with 60,000 machines and without remote working users. However, there are some points which can be

drawn from our results:

1. we limited ourselves to a simulation time period of one week. The variations of this time period make the execution time of a simulation evolve linearly. If we wanted to simulate this same attack over a period of one month, the execution time would be 24 hours.
2. further work is required to define new types of attacks. For the moment, we have mainly limited ourselves to attacks exploiting a local or remote vulnerability present on a system component. But it would be possible to model other actions that the attacker is likely to perform. For example, we could add a reasoning rule to represent phishing attacks.
3. we are currently limited by the execution time of the MulVAL algorithm which does not evolve linearly with the size of the system. Indeed, we can see from the results presented in Table 13 that when the number of assets in the system is multiplied by 9.32, the execution time of MulVAL is multiplied by 483. Nevertheless, we managed to control the evolution of the size of the generated attack graph so that it evolves linearly with the size of the system. Indeed, we can see that the size of the graph is multiplied by 33 when the number of assets in the system is multiplied by 10.
4. it would also be interesting to make the system modeling step easier. An Human Machine Interface (HMI) could be developed to make it easier to model the system and then convert this modeling into literals for MulVAL. This HMI could also make it easier for all the players responsible for IT security to work together. The modeling should also allow the reuse of some objects to avoid having to redefine each component when analyzing a new system. It would be interesting to try to make the system modeling step fully or partially automated. Information such as firewall rules, system logs or network scan results could be retrieved automatically and used to maintain a real-time system map [93], [94].

In the next section, we will discuss the advantages and disadvantages of our solution, and how it could create new opportunities in the field of risk assessment.

III.4 Discussion

III.4.1 Advantages of the solution

The main breakthrough of our solution is the ability to analyze large systems while modeling its dynamic behavior. It is possible to represent a wide variety of systems by defining the

appropriate literals and reasoning rules. Literals allow to model all the properties of a system, whether it is an IT system, an industrial network or connected objects. For example, it would be possible to define a literal *plcInput(PLCid, NumPort, Function)* to describe a Programmable Logic Controller (PLC) input. Similarly, reasoning rules allow to represent any interactions taking place in a system. These logic tools also allow to accurately model the attacker's actions and the impact of the attacks on the system. We have enhanced these modeling tools to allow the representation of dynamic system behaviors. Temporal literals can be used to model network topology changes, vulnerability discoveries, patch application or component wipe. On the other hand, the temporal reasoning rules allow to represent the time required for the realization of the interactions in the system. Therefore, we have a set of logical tools that allow us to model very finely any computer system by taking into consideration their dynamic behaviors.

We have added a system of variables that makes it easier to model complex systems by reducing the number of initial literals to write. The modeling of complex systems is also made possible by the use of efficient algorithms able to process large volumes of data. The execution time of the MulVAL algorithm has a worst-case complexity $O(N^2 \times log(N))$, with N the number of system components. We have managed to limit the size of the attack graph generated by MulVAL. We have shown that in some cases, adding an intermediate node can reduce the number of nodes in the graph. For example, this is what we did by adding the literal *lanAccess*, which reduced the number of literals *netAccess* between machines on the same network. We also developed an algorithm to remove intermediate nodes when this reduces the size of the attack graph. The execution time of our simulation algorithm has a worst-case complexity $O([\frac{n^2}{2} + n] \times |I|)$, with n the size of the graph and $|I|$ the number of time intervals. We then carried out several benchmarks to test the real performance of our solution, before finally applying it to a use case representing a computer network of a large company. These tests allowed us to validate its ability to analyze the security of complex systems.

We have defined several metrics to quantitatively assess the impact of an attack on the system. We generate curves for each node of the attack graph showing the evolution of the probability that the node is True as a function of time. This metric allows to visualize the evolution of the risk of compromise of the different components of the system. We also compute a score that represents the time required for the attacker to reach a given probability of compromising a component. This score is then used to display a heatmap of the system in order to identify the components most affected by the attack. We propose to visualize the impact of a change in system configuration or attack conditions by displaying a curve for each node representing the difference in probability of being True between two simulations. This metric can for example be used to visualize the impact of a vulnerability on the security of the system

by comparing the simulation with and without it.

III.4.2 Identified limitations

However, the ability of our solution to analyze very large systems is limited by the execution time of the attack graph generation algorithm MulVAL. Indeed, we have noticed during the realization of our benchmarks and the test on the complex use case that the execution time of this algorithm could quickly increase in some cases. It would be interesting to check if it is not possible to group components with the same properties to reduce the number of initial literals sent to MulVAL. It would also be interesting to try to improve the performance of the attack graph size optimization algorithm so that it can be used for systems greater than 1,600 components. Finally, we could try to parallelize the execution of the Datalog program in XSB when generating the attack graph.

It is important to note that we use an approximate algorithm to calculate the compromise probabilities of the different system components. Although our solution does not give exact results, it has allowed us to analyze large systems in a reasonable amount of time. Future work could be done to measure the difference on small systems between the results of our approximate algorithm and the exact calculations. The use of approximate algorithms seems to us to be mandatory for the analysis of complex systems, whether they are sampling algorithms like those used to infer probabilities in Bayesian networks [54], or simulation algorithms like those used in [81] or in our solution. Work must also be done to make the system modeling step easier. An HMI could for example be developed to (1) propose a graphical way to model the system which is then converted into literals, (2) make it easier for all security actors to work together and (3) allow the reuse of objects used in the models. It would also be interesting to work on a way to automate totally or partially the modeling step by proposing, for example, real-time system mapping tools.

In our simulations, we currently consider that the attacker has unlimited human and budget resources. This assumption allowed us to increase the time performance of our simulation algorithm. Indeed, when several actions are possible at a time t, we consider that the attacker can parallelize their exploitation. Besides the fact that it allows to go through all the attack paths without increasing the simulation time, it also avoids having to make a choice in the actions to perform. For example, if we consider that the attacker is alone and that he cannot parallelize the actions, a choice will have to be made when traversing the attack graph and several arcs lead to different reasoning rule nodes. Future work could be done to integrate the real capabilities of the attacker during the simulation of an attack and define criteria to choose

the actions to be performed in priority.

III.4.3 Opportunities

Our solution currently allows for an accurate assessment of the system's security. But it would also be interesting to model the impact of attacks on the organization's business processes. Business processes could for example be modeled with a Business Process Management (BPM) by integrating the impact of the attack on the system and to check if the business processes are still operational. Our solution could also be extended to identify the protective measures to be put in place as a priority, taking into consideration the human and budgetary limitations of the organization. This would allow system security teams to optimize their investments to effectively reduce the risk of compromise. It would also be possible to propose an algorithm to identify the optimal attack path to achieve a compromise target in the system. This could, for example, help offensive security teams such as pentesters or red teams to achieve their compromise goals.

III.4.4 Identified risks

We would like to end with a global review of risk analysis solutions based on attack path modeling. Although these solutions allow for a very accurate analysis of how a system can be compromised by a malicious actor, the amount of information required to model all attack paths can quickly become very important, especially for large systems. Indeed, in order for the risk analysis to be consistent and for the result to be as reliable as possible, it is necessary that the modeling of the system and the attacker's capabilities be as exhaustive as possible. Therefore, the interest of this type of modeling depends greatly on the possibility of obtaining large volumes of information about the system to be analyzed. In addition, some information can be very difficult to obtain. For example, it may be difficult to identify whether the password associated with an administration account is strong enough to resist a brute force attack by the attacker. Although in a defensive context, the entire system must be modeled exhaustively in order to guarantee a reliable assessment of the risk of compromise, it is sufficient for attackers to find an attack path that allows them to achieve their objectives. There is therefore a major difference in the investment required to model the system between security teams who need to be comprehensive and attackers who only need to find an attack path.

The complexity of modeling the system also comes from the uncertainty in the value of some parameters. For example, it is difficult to define the exact time required for a malicious actor to

exploit a vulnerability. In our solution, we modeled uncertainty by associating probabilities for a literal to be True over an interval of time and by representing the timing of clocks associated with reasoning rules with probability distributions. We therefore believe that the greatest difficulty in risk analysis of complex systems is to accurately model the system as well as the real capabilities of the attacker, and that this problem should be the subject of further interest.

III.5 Conclusion

In this section, we have presented the results of our work carried out to answer the research problem of this thesis. We have shown how it was possible to build an attack graph representing the attack paths present in a system by taking into account its dynamic characteristics. We have shown that taking these dynamic characteristics into account allows us to identify new attack paths that a static representation could not find.

We then proposed a solution to calculate the risk of compromise for each component of the system. The calculations are based on the simulation of several attacks performed from the previously constructed dynamic attack graph.

Then we demonstrated that the solution proposed in this thesis was able to analyze security in complex systems. We evaluated the worst-case time complexity of all the algorithms we used. Several benchmarks have been performed to evaluate the real complexity of these algorithms. Finally, we applied our solution on our dataset representing the architecture of an IT system of a large organization. The results obtained confirm that our model can be used in systems made up of several thousands of components.

Finally, we have discussed the scientific challenges addressed by our approach as well as the limitations identified. We have also discussed the scientific challenges that have been addressed by our approach as well as the limitations that remain to be solved in order to fully address the research problem of this thesis.

Chapter IV

Conclusion

Risk analysis in complex computer systems is currently a serious challenge. These systems have become ubiquitous in our modern societies and are increasingly used in critical systems, such as in the management of nuclear power plants or in weapons systems. The current methods used, such as EBIOS in France or the one proposed by the NIST in the USA, are not adapted to the high complexity of modern systems. Indeed, these methods rely on too much human work, making the analysis subjective and error-prone.

In this dissertation, we analyzed and compared several research works that provided a partial answer to this problem. Most of these solutions are based on modeling the attack paths that allow malicious actors to achieve their compromise objectives. But these models are based on a static representation of the system to be analyzed and therefore omit some dynamic properties. This can have an impact on the identification of the attack paths present in the system as we have shown in our work.

Some research works have proposed solutions to calculate the attacker's chances of success based on an attack graph and the difficulty of exploiting the vulnerabilities. Nevertheless, the calculations performed are rarely adapted to complex systems and do not allow to obtain results in a reasonable amount of time.

Several work are interested to the problem of modeling a complex system. However, the information available about the properties of the system and its components is often difficult to obtain as the system is large and complex. Some of this information may also be uncertain or unknown by nature, such as how long it will take for the attacker to exploit a vulnerability.

Our objective was therefore to propose a risk analysis solution that would overcome the limitations previously defined, that is to model the attack paths from a dynamic representation of the system based on a limited and uncertain source of information, as well as to calculate the risk of compromise in a reasonable amount of time despite the large size of the system to be analyzed. We also aimed to simplify the system modeling step as much as possible so that this task would be feasible despite its high complexity.

To achieve this we perform a set of studies on existing solutions in the literature. Our work takes advantage of previously studied solutions. We designed and developed a solution to model the attack paths from dynamic description of the system. The proposed solution is based on MulVAl which allows to generate logic-based attack graphs. We have enhanced this model by defining new logical tools able to represent the dynamic properties of systems. The temporal literals allow to represent the changes of properties of the system over time, while the temporal reasoning rules allow to represent the time required for the realization of the interactions that take place in the system.

We also simplified the system modeling step by developing a system of variables that can be used in the parameters of the literals and that significantly reduces their number. We have set up a way to represent the uncertainty related to some information: probabilities defined on time intervals are associated to literals and allow to model the evolution of the likelihood of a system property over time. Probability distributions have been used to model the activation times of the reasoning rules, which allows representing the uncertainty about this value.

Then we redeveloped the attack graph generation algorithm to take into account these new logical tools and to list all the attack paths from a dynamic description of the system.

In a second step, our work has allowed us to propose a method to assess the risk of compromise. This method consists in performing several simulations of an attack from the dynamic attack graph previously built. The result of these simulations allows us to approximate the probability of compromise of each component of the system as a function of time.

We have also proposed a method to measure the impact of one or a set of vulnerabilities on the security of the system. To do this, we calculate for each component of the system the difference in risk of compromise between two simulations: the first simulation is performed from a representation of the system including all the vulnerabilities, and the second from a representation excluding the vulnerabilities for which we want to assess the impact.

Finally, our work consisted in proving that our solution could be used in complex systems composed of several thousands of elements. We started by assessing the worst-case time complexity of the algorithms used. Then we performed a series of benchmarks to estimate the real performance of our algorithms. These tests showed that the average time complexity was linear for all our algorithms, except for the attack graph construction algorithm which is quadratic. Lastly, we tested our solution on a complex system representing the IT network of a large company composed of thousands of elements.

The purpose of this thesis was to address the requirements defined in the introduction, which are as follows:

1. the method must be applicable in any kind of computer system;
2. the method must be able to be used in complex and dynamic systems. In this thesis, we set ourselves the objective of analyzing the security of systems composed of several thousands of elements;
3. the calculations made to assess risk must be as objective as possible. To do this, the security analysis must be based on real information directly issued from the studied system;

4. the risk assessment process should be automated as much as possible to reduce the amount of work that needs to be done by humans and to avoid errors;
5. the risk analysis method must also be able to assess the impact of one or a set of vulnerabilities on the system.

Requirement 1 has been met because the method used to generate the attack graph is an abstract method capable of modeling all types of systems. We only need to define the literals and the reasoning rules that represent the system to be analyzed. The computations performed subsequently from the attack graph to evaluate the security of the system do not depend on the nature of the literals and the reasoning rules defined previously. Our method can therefore be used in any kind of computer system, whether it is an IT network or a CPS.

Requirement 2 is partially addressed. Our model is able to represent the dynamic behavior of the studied system. We have performed numerous performance tests, and in particular we have applied our solution on an IT system composed of several thousands elements. These tests show that our method is applicable in complex systems. But the non-linear evolution of the generation time of the attack graph prevents us from analyzing systems whose size is greater than several hundreds thousands of elements.

Requirement 3 is partially addressed. The attack graph generation algorithm allows to list exhaustively all the attack paths present in the system. However, compromise risk calculations are always based on subjective information. For example, the time required to exploit a vulnerability is based on its CVSS score. The metrics used to calculate this score are based on a subjective assessment by experts and it is not uncommon to find different CVSS score for the same vulnerability. However, to overcome this problem as well as the sometimes unreliable information about the properties of the system, we have adapted our model to represent this uncertainty.

Requirement 4 is partially addressed. Although the construction of the attack graph as well as the calculation of the risk of compromise is fully automated, the modeling of the system is not. We have not been able to propose an algorithm to automatically retrieve this information from the real system because it strongly depends on the nature and composition of the studied system. For an IT network, it would be possible to retrieve this information from the filtering rules present on the firewalls or to perform network scans to obtain a matrix of access rights between the system components. Some of the vulnerabilities can also be obtained automatically by performing security scans with Nessus or OpenVAS for example.

Requirement 5 is partially addressed. Our solution allows to assess the impact of one or

a set of vulnerabilities by comparing the result of attack simulations with and without them. However, the measure of the impact remains limited to the risk of compromise of the system components.

Future work should be done to remove the various limitations that we have highlighted. We should seek to improve the performance of the attack graph generation algorithm in order to analyze increasingly complex systems. The system modeling stage should be further automated. For each kind of computer system, a solution should be developed to automatically retrieve the system properties and convert them into literals and reasoning rules. Finally, the impact of an attack should be assessed at the organizational level. To do this, missions and business processes must be modeled by integrating the impact of attacks at the system level.

Appendix

Appendix A. Definition of literals and reasoning rules used in the MulVAL framework

Literal	Description
Initial literal	
networkServiceInfo(Host, Vlan, Program, Protocol, Port, User)	Indicates that service *Program* is running on machine *Host* with rights *User* and is listening on port *Port* of interface *Vlan* using protocol *Protocol*
vulExists(Host, VulID, Program, Range, Consequence)	Indicates that the vulnerability *VulID* is present on the program *Program* of the machine *Host*, can be exploited with the conditions *Range* and has as consequences *Consequence*
firewallRule(HostS, Vlan1, HostD, Vlan2, Protocol, Port)	Indicates that a firewall rule authorizes the machine *HostS* connected on the network *Vlan1* to communicate towards the machine *HostD* connected on the network *Vlan2* on the port *Protocol* with the protocol *Port*
vlanInterface(Host, Vlan)	Indicates that the machine *Host* is connected to the network *Vlan*
attackerLocated(Vlan)	Indicates that the attacker is present on the network *Vlan*
softwareInfo(Host, Software, Perm)	Indicates that software *Software* is installed on machine *Host* and can be run with user rights *Perm*
isProxy(Proxy, PortS, Srv, PortD)	Indicates that the proxy *Proxy* listens on port *PortS* and redirects all traffic to port *PortD* of the machine *Srv*
receivePhishingMail(Host)	Indicates that a malicious e-mail has been sent to the mailbox of the machine *Host*
isDC(DC, Domain)	Indicates that the domain controller *DC* is responsible for the domain *Domain*
networkStream(HostS, HostD, Port, Protocol, Vlan, Data)	Indicates that a connection is ongoing between the machine *HostS* and the machine *HostD* on the port *Port* with the protocol *Protocol*, and the data *Data* are transiting on the network *Vlan*.
openMaliciousFile(Host, User, Perm, Software)	Indicates that user *User* has opened a malicious file on machine *Host* with software *Software* and rights *Perm*
mitm(HostS, HostD, Vlan)	Indicates that it is possible to intercept communications between the machine *HostS* and the machine *HostD* on the network *Vlan*
noCheckAuth(Host, Port, Prot)	Indicates that communications to machine *Host* on port *Port* with protocol *prot* do not require authentication
isDomainMember(Host, DC, Domain)	Indicates that the machine *Host* is part of the domain *Domain* managed by the domain controller *DC*
Derived literal	
execCode(Host, User)	Indicates that the attacker is able to execute code on the machine *Host* with the rights *User*
availability(Host, Software)	Indicates that the attacker is able to make program *Software* unavailable on machine *Host*
confidentiality(Data)	Indicates that the attacker is capable of compromising the confidentiality of data *Data*
ddos(Host, Software)	Indicates that the attacker is able to launch a distributed denial of service attack on the service *Software* of the machine *Host*
netAccess(Machine, Protocol, Port)	Indicates that the attacker has network access to machine *Machine* on port *Port* with protocol *Protocol*
lanAccess(Vlan)	Indicates that the attacker has access to the network *Vlan*

Table 14: Description of the literals used in the MulVAL framework

Reasoning rule	Description
execCode	
execCode(HostD, Perm) :- vulExists(HostD, _, Software, remoteExploit, privEscalation), networkServiceInfo(HostD, _, Software, Protocol, Port, Perm), netAccess(HostD, Protocol, Port)	Indicates that the attacker can execute code on machine *HostD* with rights *Perm* if a remotely exploitable vulnerability of type Elevation of privileges is present on service *Software* of machine *HostD*, that this service is running and listening on port *Port* with protocol *Protocol*, and that the attacker has access to this service
execCode(HostD, Perm2) :- vulExists(HostD, _, Software, localExploit, privEscalation), softwareInfo(HostD, Software, Perm2), execCode(HostD, Perm1)	Indicates that the attacker can execute code on machine *HostD* with privileges *Perm2* if a locally exploitable vulnerability of type Elevation of privileges is present on the program *Software* of machine *HostD*, that this program is running with privileges *Perm2*, and that the attacker already has control over machine *HostD* with privileges *Perm1*
execCode(HostD, Perm) :- softwareInfo(HostD, Software, _), receivePhishingMail(HostD), openMaliciousFile(HostD, _, Perm, Software)	Indicates that the attacker can execute code on machine *HostD* with privileges *Perm* if the user opens a malicious file received by mail on machine *HostD* with program *Software* running with privileges *Perm*
execCode(Host, root) :- execCode(DC, root), isDC(DC, Domain), isDomainMember(Host, DC, Domain)	Indicates that the attacker can execute code on machine *Host* with privileges *root* if machine *Host* belongs to the domain *Domain* managed by the domain controller *DC* and the attacker is already executing code on the domain controller *DC* with root privileges
availability	
availability(HostD, Software) :- vulExists(HostD, _, Software, remoteExploit, dos), networkServiceInfo(HostD, _, Software, Protocol, Port, Perm), netAccess(HostD, Protocol, Port)	Indicates that the attacker can make service *Software* unavailable on machine *HostD* if a remotely exploitable vulnerability of type dos is present on service *Software* of machine *HostD*, that this service is running and listening on port *Port* with protocol *Protocol*, and that the attacker has access to this service
ddos	
ddos(HostD, Software) :- vlanInterface(HostD, 'internet'), networkServiceInfo(HostD, 'internet', Software, _, _, _)	Indicates that the attacker can launch a distributed denial of service attack on the service *Software* present on the machine *HostD* if this service is accessible from Internet
confidentiality	
confidentiality(Data) :- networkStream(HostS, HostD, Port, Protocol, Vlan, Data), noCheckAuth(HostD, Port, Protocol), mitm(HostS, HostD, Vlan)	Indicates that the attacker can compromise the confidentiality of data *Data* if the attacker can intercept a network flow on the network *Vlan* because no authentication is required

netAccess	
netAccess(HostD, Protocol, Port) :- execCode(HostS, Perm), firewallRule(HostS, Vlan1, HostD, Vlan2, Protocol, Port), vlanInterface(HostS, Vlan1), vlanInterface(HostD, Vlan2)	Indicates that the attacker has access to machine *HostD* on port *Port* with protocol *Protocol* if he executes code on a machine *HostS* connected to network *Vlan1* and that a firewall rule allows connections from network *Vlan1* to access machine *HostD* connected to network *Vlan2* on port *Port* with protocol *Protocol*
netAccess(HostD, Protocol, Port) :- attackerLocated(Vlan1), firewallRule(_, Vlan1, HostD, _, Protocol, Port), vlanInterface(HostD, _)	Indicates that the attacker has access to machine *HostD* on port *Port* with protocol *Protocol* if he has access to network *Vlan1* and that a firewall rule allows connections from network *Vlan1* to machine *HostD* on port *Port* with protocol *Protocol*
netAccess(Host, Protocol, Port) :- attackerLocated(Vlan), vlanInterface(Host, Vlan)	Indicates that the attacker has access to machine *Host* on port *Port* with protocol *Protocol* if the attacker has access to network *Vlan* and machine *Host* is connected on the same network
netAccess(HostD, Protocol, PortD) :- execCode(HostS, Perm), vlanInterface(HostS, Vlan1), isProxy(Proxy, PortP, HostD, PortD), firewallRule(HostS, Vlan1, Proxy, _, _, PortP), firewallRule(Proxy, _, HostD, _, Protocol, PortD)	Indicates that the attacker has access to machine *HostD* on port *PortD* with protocol *Protocol* if the attacker executes code on a machine *HostS* connected to network *Vlan1*, that a firewall rule authorizes connections from network *Vlan1* to proxy *Proxy* on port *PortP* and that another firewall rule authorizes connections from proxy to machine *HostD* on port *PortD* with protocol *Protocol*
netAccess(HostD, Protocol, PortD) :- attackerLocated(Vlan1), isProxy(Proxy, PortP, HostD, PortD), firewallRule(HostS, Vlan1, Proxy, _, _, PortP), firewallRule(Proxy, _, HostD, _, Protocol, PortD)	Indicates that the attacker has access to machine *HostD* on port *PortD* with protocol *Protocol* if the attacker has access to network *Vlan1*, that a firewall rule authorizes connections from network *Vlan1* to proxy *Proxy* on port *PortP* and that another firewall rule authorizes connections from proxy to machine *HostD* on port *PortD* with protocol *Protocol*
netAccess(HostD, Protocol, Port) :- execCode(HostS, Perm), firewallRule(HostS, internet, HostD, Vlan2, Protocol, Port), vlanInterface(HostS, homeNetwork), vlanInterface(HostD, Vlan2)	Indicates that the attacker has access to machine *HostD* on port *Port* with protocol *Protocol* if he executes code on a machine *HostS* that is connected to a home network and that a firewall rule allows connections from the Internet to access a machine *HostD* on port *Port* with protocol *Protocol*
netAccess(HostD, Protocol, Port) :- lanAccess(Vlan), vlanInterface(HostD, Vlan)	Indicates that the attacker has access to machine *HostD* on port *Port* with protocol *Protocol* if he has access to network *Vlan* and machine *HostD* is connected on this network.
lanAccess	
lanAccess(Vlan) :- execCode(Host, Perm), vlanInterface(Host, Vlan)	Indicates that the attacker has access to the network *Vlan* if he is already executing code on a machine *Host* and this machine is connected to the network *Vlan*

Table 15: Description of the reasoning rules used in the MulVAL framework

Appendix B. Description of the assets and vulnerabilities present in the use case of the complex computer network

CVE ID	Asset	Description	CVSSv3: Attack Complexity (AC), CVSSv2: Access Complexity (AC)	CVSSv3: Privileges Required (PR)	CVSSv3: User Interaction (UI)
				CVSSv2: Authentication (AU)	
CVE-2010-0425	Internal firewalls	Remote code execution	Low (0.71)	None	
CVE-2008-4609	Switch	Denial of service	Medium (0.61)	None	
CVE-2021-22992	Proxy BIG-IP	Denial of service	Low (0.77)	None (0.85)	None (0.85)
CVE-2021-22986	Proxy BIG-IP	Remote code execution	Low (0.77)	None (0.85)	None (0.85)
CVE-2016-8740	SRV_APP_01, SRV_APP_02, SRV_WEB_01	Denial of service	Low (0.77)	None (0.85)	None (0.85)
CVE-2019-3855	SRV_APP_02, SRV_APP_03, SRV_APP_04, SRV_WEB_01	Remote code execution	Low (0.77)	None (0.85)	Required (0.62)
CVE-2019-0199	SRV_APP_01	Denial of service	Low (0.77)	None (0.85)	None (0.85)
CVE-2020-1938	SRV_APP_02	Remote code execution	Low (0.77)	None (0.85)	None (0.85)
CVE-2020-9484	SRV_APP_02	Remote code execution	High (0.44)	Low (0.62)	None (0.85)
CVE-2020-11996	SRV_APP_01	Denial of service	Low (0.71)	None (0.85)	None (0.85)
CVE-2020-13934	SRV_APP_01	Denial of service	Low (0.71)	None (0.85)	None (0.85)
CVE-2021-30639	SRV_APP_01	Denial of service	Low (0.71)	None (0.85)	None (0.85)
CVE-2021-34473	SRV_MAIL_01	Remote code execution	Low (0.77)	None (0.85)	None (0.85)
CVE-2019-12827	SRV_IPBX	Denial of service	Low (0.71)	Low (0.62)	None (0.85)
CVE-2019-13161	SRV_IPBX	Denial of service	High (0.44)	Low (0.62)	None (0.85)
CVE-2019-18790	SRV_IPBX	Denial of service	Low (0.71)	None (0.85)	None (0.85)

CVE-2020-35652	SRV_IPBX	Denial of service	Low (0.71)	Low (0.62)	None (0.85)
CVE-2021-26717	SRV_IPBX	Denial of service	Low (0.71)	None (0.85)	None (0.85)
CVE-2020-35776	SRV_IPBX	Denial of service	Low (0.71)	None (0.85)	Required (0.62)
CVE-2021-26906	SRV_IPBX	Denial of service	High (0.44)	None (0.85)	None (0.85)
CVE-2019-15297	SRV_IPBX	Denial of service	Low (0.71)	None (0.85)	None (0.85)
CVE-2021-31878	SRV_IPBX	Denial of service	Low (0.71)	Low (0.62)	None (0.85)
CVE-2021-32686	SRV_IPBX	Denial of service	High (0.44)	None (0.85)	None (0.85)
CVE-2020-0796	PF_WIN7, PP_WIN7, PF_XP, PP_XP	Remote code execution	Low (0.71)	None (0.85)	None (0.85)

Table 16: Characteristics of the vulnerabilities present in the complex use case network

Asset	Description	Count
FW-XX-INT	Internal firewall, one for each site of the company	5
FW-XX-EXT	External firewall, one for each site of the company	5
SWO-XX-YY	Cisco switches used at each site to connect workstations to the corporate network	135
SWS-XX-YY	Cisco switches used at each site to connect servers to the corporate network	15
SRV-APP-YY	Enterprise application servers	10
SRV-WEB-YY	Company's web server	3
SRV-BDD-YY	Enterprise database servers	3
SRV-MAIL-01	Company's mail server	1
SRV-XX-IPBX	Company IP telephony server, one for each site	5
ADDC-XX	Domain controller servers, one for each company site	5
PF-W10-YY	Desktop workstations equipped with the Windows 10 operating system, deployed on the different sites of the company	5,000
PF-W7-YY	Desktop workstations equipped with the Windows 7 operating system, deployed on the different sites of the company	5,000
PF-XP-YY	Desktop workstations equipped with the Windows XP operating system, deployed on the different sites of the company	5,000
PP-W10-YY	Laptop workstations equipped with the Windows 10 operating system, deployed on the different sites of the company	5,000
PP-W7-YY	Laptop workstations equipped with the Windows 7 operating system, deployed on the different sites of the company	5,000
PP-XP-YY	Laptop workstations equipped with the Windows XP operating system, deployed on the different sites of the company	5,000
IPC-YY	IP phones distributed throughout the company's sites	30,000
IMP-XX-YY	HP printers present on all the sites of the company	120
AP-XX	Access point Wifi present in the lobby of each site of the organization	5

Table 17: List of assets present in the network of the complex use case ($XX \in \{PA, ST, MA, LY, BR\}$, $YY \in \mathbb{N}$)

References

[1] "CrowdStrike's work with the Democratic National Committee: Setting the record straight." (), [Online]. Available: https://www.crowdstrike.com/blog/bears-midst-intrusion-democratic-national-committee/.

[2] L. Skyttner, *General Systems Theory: An Introduction*. 1996.

[3] J. Ladyman, K. Wiesner, and J. Lambert, "What is a complex system?" *European Journal for Philosophy of Science*, vol. 3, pp. 33–67, 2013.

[4] "MITRE ATT&CK." (), [Online]. Available: https://attack.mitre.org/.

[5] D. A. Fernandes, L. F. Soares, J. V. Gomes, M. M. Freire, and P. R. Inácio, "Chapter 25 - A Quick Perspective on the Current State in Cybersecurity," in *Emerging Trends in ICT Security*, B. Akhgar and H. R. Arabnia, Eds. 2014, pp. 423–442.

[6] M. D. L. Maxat Akbanov Vassilios G. Vassilakis, "WannaCry Ransomware: Analysis of Infection, Persistence, Recovery Prevention and Propagation Mechanisms," *Journal of Telecommunications and Information Technology*, pp. 113–124, 2019.

[7] E. D. Wolff, K. M. Growley, M. O. Lerner, M. B. Welling, M. G. Gruden, and J. Canter, "Navigating the SolarWinds Supply Chain Attack," *The Procurement Lawyer*, vol. 56, no. 2, 2021.

[8] "Advanced Persistent Threats: Learn the ABCs of APTs - Part A." (), [Online]. Available: https://www.secureworks.com/blog/advanced-persistent-threats-apt-a.

[9] J. E. Sullivan and D. Kamensky, "How cyber-attacks in Ukraine show the vulnerability of the U.S. power grid," *The Electricity Journal*, vol. 30, no. 3, pp. 30–35, 2017.

[10] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet Dossier," Symantec Security Response, Tech. Rep., 2011.

[11] *Nessus scanner*. [Online]. Available: https://www.tenable.com/products/nessus.

[12] "La méthode EBIOS Risk Manager." (), [Online]. Available: https://www.ssi.gouv.fr/entreprise/management-du-risque/la-methode-ebios-risk-manager/.

[13] "NIST Risk Management Framework (RMF)." (), [Online]. Available: https://csrc.nist.gov/projects/risk-management/about-rmf.

[14] "Guide for Conducting Risk Assessments." (), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf.

[15] E. Doyle, D. McGovern, and S. McCarthy, "Compliance-Innovation Enabling strategic growth White Paper April 13," Nov. 2013.

[16] M. M. H. Onik, C.-S. KIM, and J. Yang, “Personal Data Privacy Challenges of the Fourth Industrial Revolution,” Feb. 2019, pp. 635–638. DOI: 10.23919/ICACT.2019.8701932.

[17] G. Liang and W. Li, “A novel industrial control architecture based on Software-Defined Network,” *Measurement and Control*, vol. 51, p. 002 029 401 878 431, Jul. 2018. DOI: 10.1177/0020294018784310.

[18] “A full range of high-performance systems.” (), [Online]. Available: https://www.naval-group.com/en/systems.

[19] V. Nagaraju, L. Fiondella, and T. Wandji, “A survey of fault and attack tree modeling and analysis for cyber risk management,” *IEEE Conference on Technologies for Homeland Security*, 2017.

[20] M. Audinot, S. Pinchinat, and B. Kordy, “Guided Design of Attack Trees: A System-Based Approach,” *Computer Security Foundations Workshop*, pp. 61–75, 2018.

[21] A. E. M. AL-Dahasi and B. N. A. Saqib, “Attack tree Model for Potential Attacks Against the SCADA System,” *Telecommunications Forum (TELFOR)*, 2019.

[22] R. Maciel, J. Araujo, J. Dantas, C. Melo, E. Guedes, and P. Maciel, “Impact of a DDoS attack on computer systems: An approach based on an attack tree model,” *Annual IEEE Systems Conference*, 2018.

[23] H.-K. Kong, M. K. Hong, and T.-S. Kim, “Security risk assessment framework for smart car using the attack tree analysis,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, pp. 531–551, 2018.

[24] R. W. Ritchey and P. Ammann, “Using Model Checking to Analyze Network Vulnerabilities,” *IEEE Symposium on Security and Privacy*, pp. 156–165, 2000.

[25] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, “Automated Generation and Analysis of Attack Graphs,” *IEEE Symposium on Security and Privacy*, pp. 273–284, 2002.

[26] P. Ammann, D. Wijesekera, and S. Kaushik, “Scalable, Graph-Based Network Vulnerability Analysis,” *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002.

[27] M. S. Barik, A. Sengupta, and C. Mazumdar, “Attack Graph Generation and Analysis Techniques,” *Defence Science Journal*, vol. 66, no. 6, pp. 559–567, 2016.

[28] S. Jajodia and S. Noel, “Topological Vulnerability Analysis: A Powerful New Approach For Network Attack Prevention, Detection, and Response,” in *Statistical Science and Interdisciplinary Research*, S. K. Pal, Ed. 2008, vol. 3, pp. 285–305.

[29] S. Jajodia, S. Noel, and B. O’Berry, “Topological Analysis of Network Attack Vulnerability,” in *Managing Cyber Threats*, V. Kumar, J. Srivastava, and A. Lazarevic, Eds. 2005, pp. 247–266.

[30] *Retina IoT (RIoT)*. [Online]. Available: https://www.beyondtrust.com/press/offers-free-cloud-based-enterprise-iot-vulnerability-scanner.

[31] X. Ou, S. Govindavajhala, and A. W. Appel, “MulVAL: A Logic-based Network Security Analyzer,” in *Proceedings of the 14th, USENIX, Security Symposium*, 2005.

[32] S. Ceri, G. Gottlob, and L. Tanca, “What you Always Wanted to Know About Datalog (And Never Dared to Ask),” *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 146–166, 1989.

[33] K. Sagonas, T. Swift, and D. S. Warren, “XSB as an Efficient Deductive Database Engine,” *ACM SIGMOD Record*, vol. 23, no. 2, 1999.

[34] X. Ou, W. F. Boyer, and M. A. McQueen, “A Scalable Approach to Attack Graph Generation,” *Conference on Computer and Communications Security*, pp. 336–345, 2006.

[35] K. Ingols, R. Lippmann, and K. Piwowarski, “Practical Attack Graph Generation for Network Defense,” *ACSAC’06*, IEEE, Ed., pp. 121–130, 2006.

[36] B. Yiğit, G. Gür, F. Alagöz, and B. Tellenbach, “Cost-aware securing of IoT systems using attack graphs,” *Ad Hoc Networks*, vol. 86, pp. 23–35, 2019.

[37] L. Wang, A. Singhal, and S. Jajodia, “Measuring the Overall Security of Network Configurations Using Attack Graphs,” *IFIP Annual Conference on Data and Applications Security and Privacy*, pp. 98–112, 2007.

[38] K. Bi, D. Han, and J. Wang, “K maximum probability attack paths dynamic generation algorithm,” *Computer Science and Information Systems*, vol. 13, no. 2, pp. 677–689, 2016.

[39] F. Dai, Y. Hu, K. Zheng, and B. Wu, “Exploring risk flow attack graph for security risk assessment,” *IET Information Security*, vol. 9, no. 6, pp. 344–353, 2015.

[40] U. Garg, G. Sikka, and L. K. Awasthi, “Empirical analysis of attack graphs for mitigating critical paths and vulnerabilities,” *Computers & Security*, vol. 77, pp. 349–359, 2018.

[41] N. Polatidis, M. Pavlidis, and H. Mouratidis, “Cyber-attack path discovery in a dynamic supply chain maritime risk management system,” *Computer Standards & Interfaces*, vol. 56, pp. 74–82, 2018.

[42] B. Sultan, F. Dagnat, and C. Fontaine, “A Methodology to Assess Vulnerabilities and Countermeasures Impact on the Missions of a Naval System,” *Computer Security*, pp. 63–76, 2017.

[43] P. Johnson, A. Vernotte, M. Ekstedt, and R. Lagerström, “pwnPr3d: an Attack-Graph-Driven Probabilistic Threat-Modeling Approach,” *11th International Conference on Availability, Reliability and Security (ARES)*, pp. 278–283, 2016.

[44] Y. Yun, X. Xi-shan, and Q. Zhi-chang, “A Probabilistic Computing Approach of Attack Graph-Based Nodes in Large-Scale Network,” *Procedia Environmental Sciences*, vol. 10, pp. 3–8, 2011.

[45] A. Singhal and X. Ou, “Security Risk Analysis of Enterprise Networks Using Probabilistic Attack Graphs,” *Network Security Metrics*, pp. 53–73, 2017.

[46] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, “An Attack Graph-Based Probabilistic Security Metric,” in *Data and Applications Security XXII*, V. Atluri, Ed. 2008, pp. 283–296.

[47] J. Homer, X. Ou, and D. Schmidt, *A Sound and Practical Approach to Quantifying Security Risk in Enterprise Networks*, Technical Reports from Department of Computing and Information Sciences, Kansas State University, 2009.

[48] *OMG Meta Object Facility (MOF) Core Specification*. [Online]. Available: `https://www.omg.org/spec/MOF/2.5.1/PDF`.

[49] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, “Section 24.3: Dijkstra’s algorithm,” *Introduction to Algorithms*, pp. 595–601, Jan. 2001.

[50] N. Juha-Pekka, K. Ilmari, and V. Janne, “BAYESIAN NETWORKS – AN EXAMPLE OF SOFTWARE AND SOME DEFENCE APPLICATIONS,” *(R)evolution of War*, 2015.

[51] L. Muñoz-González, D. Sgandurra, M. Barrère, and E. C. Lupu, “Exact Inference Techniques for the Analysis of Bayesian Attack Graphs,” *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 231–244, 2019.

[52] J. Pearl, “Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach,” in *Probabilistic and Causal Inference: The Works of Judea Pearl*. 2022, pp. 129–138.

[53] M. Vasimuddin, S. P. Chockalingam, and S. Aluru, “A Parallel Algorithm for Bayesian Network Inference using Arithmetic Circuits,” *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 34–43, 2018.

[54] L. Muñoz-González, D. Sgandurra, A. Paudice, and E. C. Lupu, “Efficient Attack Graph Analysis through Approximate Inference,” *ACM Transactions on Privacy and Security*, vol. 20, no. 3, 2016.

[55] K. W. Afrassa, A. Z. Boz, M. F. Amasyali, and S. Tahar, “Benchmarking BNT Inference Engines using an Early Warning System,” *Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2020.

[56] “Inference in Bayesian networks.” (), [Online]. Available: `https://courses.csail.mit.edu/6.034s/handouts/spring12/chapter14_mod_b.pdf`.

[57] S. Geman and D. Geman, “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[58] N. Poolsappasit, R. Dewri, and I. Ray, “Dynamic Security Risk Management Using Bayesian Attack Graphs,” *Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012.

[59] X. Li, M. Li, and H. Wang, “Research on Network Security Risk Assessment Method Based on Bayesian Reasoning,” *International Conference on Electronics Information and Emergency Communication*, 2019.

[60] J. Sembiring, M. Ramadhan, Y. S. Gondokaryono, and A. A. Arman, “Network Security Risk Analysis using Improved MulVAL Bayesian Attack Graphs,” *International Journal on Electrical Engineering and Informatics*, vol. 7, no. 4, pp. 735–753, 2015.

[61] P. Xie, J. H. Li, X. Ou, P. Liu, and R. Levy, “Using Bayesian networks for cyber security analysis,” *International Conference on Dependable Systems and Networks (DSN)*, 2010.

[62] Q. Zhang, C. Zhou, Y.-C. Tian, N. Xiong, Y. Qin, and B. Hu, “A Fuzzy Probability Bayesian Network Approach for Dynamic Cybersecurity Risk Assessment in Industrial Control Systems,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2497–2506, 2018.

[63] A. Behfarnia and A. Eslami, “Risk Assessment of Autonomous Vehicles Using Bayesian Defense Graphs,” *IEEE 88th Vehicular Technology Conference (VTC-Fall)*, 2018.

[64] M. Frigault, L. Wang, A. Singhal, and S. Jajodia, “Measuring Network Security Using Dynamic Bayesian Network,” *Conference on Computer and Communications Security*, pp. 23–30, 2008.

[65] S. Chockalingam, W. Pieters, A. Teixeira, and P. van Gelder, “Bayesian Network Models in Cyber Security: A Systematic Review,” *Nordic Conference on Secure IT Systems*, pp. 105–122, 2017.

[66] M. J. Pappaterra and F. Flammini, “A Review of Intelligent Cybersecurity with Bayesian Networks,” *IEEE International Conference on Systems, Man and Cybernetics*, pp. 445–452, 2019.

[67] S. E. Shimony and J. Eugene Santos, “Exploiting Case-Based Independence for Approximating Marginal Probabilities,” *International Journal of Approximate Reasoning*, vol. 14, no. 1, pp. 25–54, 1996.

[68] "Petri nets," Université Paris Saclay. [Online]. Available: http://www.lsv.fr/~schwoon/enseignement/verification/ws1617/nets.pdf.

[69] Y. Xu and R. Fu, "Petri Net-based Power CPS Network Attack and Impact Modeling," *International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pp. 1107–1110, 2018.

[70] M. H.Henry, R. M.Layer, and D. R.Zaret, "Coupled Petri nets for computer network risk analysis," *International Journal of Critical Infrastructure Protection*, vol. 3, no. 2, pp. 67–75, 2010.

[71] X. Liu, J. Zhang, and P. Zhu, "Modeling cyber-physical attacks based on probabilistic colored Petri nets and mixed-strategy game theory," *International Journal of Critical Infrastructure Protection*, vol. 16, pp. 13–25, 2017.

[72] M. D. Traore, H. Jin, D. Zou, W. Qiang, and G. Xiang, "RAPn: Network Attack Prediction Using Ranking Access Petri Net," *ChinaGrid Annual Conference (ChinaGrid)*, pp. 108–115, 2011.

[73] T. M. Chen, J. C. Sanchez-Aarnoutse, and J. Buford, "Petri Net Modeling of Cyber-Physical Attacks on Smart Grid," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 741–749, 2011.

[74] M. Szpyrka and B. Jasiul, "Evaluation of Cyber Security and Modelling of Risk Propagation with Petri Nets," *Symmetry*, vol. 9, no. 3, 2017.

[75] L. M. Almutairi and S. Shetty, "Generalized Stochastic Petri Net Model Based Security Risk Assessment of Software Defined Networks," *MILCOM*, IEEE, Ed., pp. 545–550, 2017.

[76] W. Knottenbelt, "PIPE v2.5: A Petri net tool for performance modelling," *Proc. 23rd Latin American Conference on Informatics (CLEI'07)*, 2007.

[77] R. Wu, W. Li, and H. Huang, "An Attack Modeling Based on Hierarchical Colored Petri Nets," *International Conference on Computer and Electrical Engineering*, pp. 918–921, 2008.

[78] M. López, A. Peinado, and A. Ortiz, "An extensive validation of a SIR epidemic model to study the propagation of jamming attacks against IoT wireless networks," *Computer Networks*, 2019.

[79] D. D. Guglielmo, S. Brienza, and G. Anastasi, "IEEE 802.15.4e: A survey," *Computer Communications*, vol. 88, pp. 1–24, 2016.

[80] C. Del-Valle-Soto, C. Mex-Perera, R. Monroy, and J. A. Nolazco-Flores, "On the Routing Protocol Influence on the Resilience of Wireless Sensor Networks to Jamming Attacks," *Special Issue "Wireless Sensor Networks and the Internet of Things"*, vol. 15, no. 4, pp. 7619–7649, 2015.

[81] S. König, S. Rass, S. Schauer, and A. Beck, "Risk Propagation Analysis and Visualization using Percolation Theory," *IJACSA*, vol. 7, no. 1, pp. 694–701, 2016.

[82] K. Christensen. "Percolation Theory." (), [Online]. Available: https://web.mit.edu/ceder/publications/Percolation.pdf.

[83] P. Erdös and A. Rényi, "On random graphs," *Publicationes Mathematicae Debrecen*, vol. 6, p. 290, 1959.

[84] R. A. Hanneman. "Introduction to social network methods." (), [Online]. Available: https://faculty.ucr.edu/~hanneman/nettext/C11_Cliques.html.

[85] W. Chunlei, M. Qing, and F. Lan, "A COMPLEX NETWORK ANALYSIS MODEL FOR CYBERSPACE SURVIVABILITY," *International Conference on Information and Network Security (ICINS)*, 2014.

[86] P. Panigrahi and somnath Maity, “Vulnerability Analysis of Weighted Indian Power Grid Network Based on Complex Network Theory,” *14th IEEE India Council International Conference (INDICON)*, 2017.

[87] A. Singh and P. K. Mishra, “Performance Analysis of Floyd Warshall Algorithm vs Rectangular Algorithm,” *International Journal of Computer Applications*, vol. 107, no. 16, 2014.

[88] P. Bellini, R. Mattolini, and P. Nesi, “Temporal Logics for Real-Time System Specification,” *ACM Computing Surveys*, vol. 32, no. 1, pp. 12–42, 2000.

[89] P. Ammann, J. Pamula, R. Ritchey, and J. Street, “A Host-Based Approach to Network Attack Chaining Analysis,” *Annual Computer Security Applications Conference*, pp. 84–94, 2005.

[90] A. Horn, “On sentences which are true of direct unions of algebras,” *Journal of Symbolic Logic*, vol. 16, no. 1, pp. 14–21, 1951.

[91] H. Holm, “A Large-Scale Study of the Time Required to Compromise a Computer System,” *Transactions on Dependable and Secure Computing*, pp. 2–15, 2014.

[92] “The Log Normal Distribution.” (), [Online]. Available: `https://web.mit.edu/~r/current/arch/amd64_linux26/lib/R/library/stats/html/Lognormal.html`.

[93] “SolarWinds Network Discovery Tool.” (), [Online]. Available: `https://www.solarwinds.com/network-performance-monitor/use-cases/network-discovery-tool?CMP=BIZ-RVW-SWTH-AssetDiscvryTlsClsfctin-NPM`.

[94] “Automated Asset Discovery.” (), [Online]. Available: `https://www.balbix.com/solutions/it-asset-discovery-inventory-management/`.

List of Abbreviations

AC: Alternating Current.

AODV: Ad-hoc On-demand Distance Vector.

APT: Advanced Persistence Threat.

ANSSI: Agence Nationale de la Sécurité des Systèmes d'Information.

BAG: Bayesian Attack Graph.

BFS: Breadth-First Search.

BPM: Business Process Management.

BYOD: Bring Your Own Device.

CPS: Cyber-Physical System.

CPT: Conditional Probability Table.

CPU: Central Processing Unit.

CVE: Common Vulnerabilities Exposures.

CVSS: Common Vulnerability Scoring System.

DBN: Dynamic Bayesian Network.

DC: Domain Controller.

DDoS: Distributed Denial of Service.

DFS: Depth-First Search.

DMZ: Demilitarized Zone.

DPID: Datapath ID.

DSR: Dynamic Source Routing.

FTP: File Transfer Protocol.

GB: Gigabyte.

GSPN: Generalized Stochastic Petri Nets.

HMI: Human-Machine Interface.

HR: Human Resources.

ICS: Industrial Control System.

IoT: Internet of Things.

IP: Internet Protocol.

IT: Information Technology.

LLDP: Link Layer Discovery Protocol.

MAC: Medium Access Control.

MOF: Meta Object Facility.

MPHP: Multi-Parent Hierarchical Protocol.

MulVAL: Multi host, multi stage Vulnerability Analysis tool.

NetSPA: Network Security Planning Architecture.

NIST: National Institute of Standards and Technology.

NVD: National Vulnerability Database.

OSI: Open Systems Interconnection.

PIPE: Platform-Independent Petri net Editor.

PLC: Programmable Logic Controller.

PoC: Proof of Concept.

RDP: Remote Desktop Protocol.

RMF: Risk Management Framework.

RPC: Remote Procedure Call.

SDN: Software-Defined Networking.

SMB: Server Message Block.

SSH: Secure Shell.

TB: Terabyte.

TTC: Time To Compromise.

TTP: Tactics, Techniques and Procedures.

TVA: Topological Vulnerability Analysis.

USA: United States of America.

USB: Universal Serial Bus.

VLAN: Virtual Local Area Network.

VPN: Virtual Private Network.

WAP: Wireless Access Point.

ECOLE DOCTORALE

Titre: Une approche basée sur les graphes d'attaque dynamiques pour l'évaluation de l'impact des vulnérabilités dans les systèmes informatiques complexes

Mots clés: Évaluation des risques, sécurité des réseaux, système dynamique, graphe d'attaque, simulation, graphe d'attaque dynamique

Résumé: De nos jours, les réseaux informatiques sont utilisés dans de nombreux domaines et leur défaillance peut avoir un fort impact sur notre vie quotidienne. L'évaluation de leur sécurité est une nécessité pour réduire le risque de compromission par un attaquant. Néanmoins, les solutions proposées jusqu'à présent sont rarement adaptées à la grande complexité des systèmes informatiques modernes. Elles reposent souvent sur un travail humain trop important et les algorithmes utilisés ne sont pas assez performants. De plus, l'évolution du système dans le temps est rarement modélisée et n'est donc pas prise en compte dans l'évaluation de sa sécurité.
Dans cette thèse, nous proposons un nouveau modèle de graphe d'attaque construit à partir d'une description dynamique du système. Nous avons mis en évidence à travers nos expériences que notre modèle permettait d'identifier davantage de chemins d'attaque qu'un modèle de graphe d'attaque statique. Nous avons ensuite proposé un algorithme de simulation d'attaques permettant d'approximer les chances de succès de compromission du système par un acteur malveillant.
Nous avons également prouvé que notre solution était capable d'analyser la sécurité de systèmes complexes. La complexité en temps dans le pire des cas a été évaluée pour chaque algorithme utilisé et plusieurs tests ont été réalisés pour mesurer leurs performances réelles. Pour terminer, nous avons appliqué notre solution sur un réseau informatique composé de plusieurs milliers d'éléments.
De futurs travaux devraient être réalisés pour améliorer les performances de l'algorithme de génération des graphes d'attaque afin de permettre d'analyser des systèmes toujours plus complexes. Des solutions devraient également être trouvées pour faciliter l'étape de modélisation du système qui reste encore à ce jour une tâche difficile à réaliser, surtout par des humains. Enfin, l'algorithme de simulation pourrait être amélioré pour être plus réaliste et tenir compte des réelles capacités de l'attaquant. Il serait également intéressant d'évaluer l'impact des attaques au niveau de l'organisation et de ses processus métiers.

Title: A Dynamic Attack Graphs based approach for Impact Assessment of Vulnerabilities in Complex Computer Systems

Keywords: risk assessment, network security, dynamic system, attack graph, simulation, dynamic attack graph

Abstract: Nowadays, computer networks are used in many fields and their breakdown can strongly impact our daily life. Assessing their security is a necessity to reduce the risk of compromise by an attacker. Nevertheless, the solutions proposed so far are rarely adapted to the high complexity of modern computer systems. They often rely on too much human work and the algorithms used don't scale well. Furthermore, the evolution of the system over time is rarely modeled and is therefore not considered in the evaluation of its security.
In this thesis, we propose a new attack graph model built from a dynamic description of the system. We have shown through our experimentations that our model allows to identify more attack paths than a static attack graph model. We then proposed an attack simulation algorithm to approximate the chances of success of system compromise by a malicious actor. We also proved that our solution was able to analyze the security of complex systems. The worst-case time complexity was assessed for each algorithm used. Several tests were performed to measure their real performances. Finally, we applied our solution on an IT network composed of several thousands elements.
Future work should be done to improve the performance of the attack graph generation algorithm in order to analyze increasingly complex systems. Solutions should also be found to facilitate the system modeling step which is still a difficult task to perform, especially by humans. Finally, the simulation algorithm could be improved to be more realistic and take into account the real capabilities of the attacker. It would also be interesting to assess the impact of the attacks on the organization and its business processes.

Institut Polytechnique de Paris

91120 Palaiseau, France