Towards Automated Network Mitigation Analysis

Patrick Speicher*, Marcel Steinmetz*, Jörg Hoffmann†, Michael Backes*, and Robert Künnemann*
*CISPA Helmholtz Center
†CISPA Helmholtz Center, Saarland University
Email: {first name}.{last name}@cispa.saarland, backes@cispa.saarland, hoffmann@cs.uni-saarland.de

ABSTRACT

Penetration testing is a well-established practical concept for the identification of potentially exploitable security weaknesses and an important component of a security audit. Providing a holistic security assessment for networks consisting of several hundreds hosts is hardly feasible though without some sort of mechanization. Mitigation, prioritizing counter-measures subject to a given budget, currently lacks a solid theoretical understanding and is hence more art than science. In this work, we propose the first approach for conducting comprehensive what-if analyses in order to reason about mitigation in a conceptually well-founded manner. To evaluate and compare mitigation strategies, we use *simulated penetration testing*, i.e., automated attack-finding, based on a network model to which a subset of a given set of mitigation actions, e.g., changes to the network topology, system updates, configuration changes etc. is applied. Using *Stackelberg planning*, we determine optimal combinations that minimize the maximal attacker success (similar to a Stackelberg game), and thus provide a well-founded basis for a holistic mitigation strategy. We show that these Stackelberg planning models can largely be derived from network scan, public vulnerability databases and manual inspection with various degrees of automation and detail, and we simulate mitigation analysis on networks of different size and vulnerability.

CCS CONCEPTS

• **Security and privacy → Economics of security and privacy**; *Formal security models*; • **Computing methodologies → Planning under uncertainty**;

KEYWORDS

Planning, network security, simulated penetration testing

ACM Reference Format:
Patrick Speicher*, Marcel Steinmetz*, Jörg Hoffmann†, Michael Backes*, and Robert Künnemann* . 2019. Towards Automated Network Mitigation Analysis. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3297280.3297473

SAC'19, April 8-12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297473

1 INTRODUCTION

Penetration testing (pentesting) evaluates the security of an IT infrastructure by trying to identify and exploit vulnerabilities. It constitutes a central, often mandatory component of a security audit, e.g., the Payment Card Industry Data Security Standard prescribes 'network vulnerability scans at least quarterly and after any significant change in the network' [6]. Network pentests are frequently conducted on networks with hundreds of machines. Here, the vulnerability of the network is a combination of host-specific weaknesses that compose to an attack. Consequently, an exhausting search is out of question, as the search space for these combinations grows exponentially with the number of hosts. Choosing the right attack vector requires a vast amount of experience, arguably making network pentesting more art than science.

While it is conceivable that an experienced analyst comes up with several of the most severe attack vectors, this is not sufficient to provide for a sound mitigation strategy, as the evaluation of a mitigation strategy requires a holistic security assessment. So far, there is no rigorous foundation for what is arguably the most important step, the step *after* the pentest: how to mitigate these vulnerabilities.

In practice, the severity of weaknesses is assessed more or less in isolation, proposed counter-measures all too often focus on single vulnerabilities, and the mitigation path is left to the customer. There are exceptions, but they require considerable manual effort.

Simulated pentesting was proposed to automate large-scale network testing by simulating the attack finding process based on a logical model of the network. The model may be generated from network scans, public vulnerability databases and manual inspection with various degrees of automation and detail. To this end, AI planning methods have been proposed [2, 22] and in fact used commercially, at a company called Core Security, since at least 2010 [7]. These approaches, which derive from earlier approaches based on attack graphs [26, 31, 32], assume complete knowledge over the network configuration, which is often unavailable to the modeller, as well as the attacker. We follow a more recent approach favouring Markov decisions processes (MDP) as the underlying state model to obtain a good middle ground between accuracy and practicality [8, 12] (we discuss this in detail as part of our related work discussion, Section 2).

Simulated pentesting has been used to great success, but an important feature was overseen so far. If a model of the network is given, one can reason about possible mitigations without implementing them – namely, by simulating the attacker on a modified model. This allows for analysing and comparing different mitigation strategies in terms of the (hypothetical) network resulting from their application. This problem was recently introduced as *Stackelberg planning* in the AI community [34]. Algorithmically, the attacker-planning problem

now becomes part of a larger what-if planning problem, in which the best mitigation plans are constructed.

Mitigation actions can represent, but are not limited to, changes to the network topology, e.g., adding a packet filter, system updates that remove vulnerabilities, and configuration changes or application-level firewalls which work around issues. The algorithm computes optimal combinations w.r.t. minimizing the maximal attacker success for a given budget, and proposes dominant mitigation strategies with respect to cost and attacker success probability.

After discussing related work in Section 2 and giving a running example in Section 3, we present the mitigation analysis model in Section 4, framed in a formalism suited for a large range of mitigation/attack planning problems. In Section 5, we show how to derive these models by scanning a given network using the Nessus network-vulnerability scanner. The attacker action model is then derived using a vulnerability database and data associated using the Common Vulnerability Scoring System (CVSS). This methodology provides a largely automated method of deriving a model (only the network topology needs to be given by hand), which can then be used as it is, or further refined. In Section 6, we evaluate our algorithms w.r.t. problems from this class, derived from a vulnerability database and a simple scalable network topology.

2 RELATED WORK

Our work is rooted in a long line of research on network security modeling and analysis, starting with the consideration of *attack graphs*. The simulated pentesting branch of this research essentially formulates attack graphs in terms of standard sequential decision making models — *attack planning* — from AI. We give a brief background on the latter first, before considering the history of attack graph models.

Automated Planning is one of the oldest sub-areas of AI (see [9] for a comprehensive introduction). The area is concerned with general-purpose planning mechanisms that automatically find a *plan*, when given as input a high-level description of the relevant world properties (the *state variables*), the *initial state*, a *goal* condition, and a set of *actions*, where each action is described in terms of a *precondition* and a *postcondition* over state variable values. In *classical planning*, the initial state is completely known and the actions are deterministic, so the underlying state model is a directed graph (the *state space*) and the plan is a path from the initial state to a goal state in that graph. In *probabilistic planning*, the initial state is completely known but the action outcomes are probabilistic, so the underlying state model is a Markov decision process (MDP) and the plan is an action *policy* mapping states to actions.

The founding motivation for Automated Planning mechanisms is flexible decision taking in autonomous systems, yet the generality of the models considered lends itself to applications as diverse as the control of modular printers [28], natural language sentence generation [16, 17], and, in particular, network security penetration testing [2, 8, 12, 22, 29].

Simulated pentesting is rooted in the consideration of attack graphs, first introduced by Philipps and Swiler [26]. An attack graph breaks down the space of possible attacks into atomic components, often referred to as attack actions, where each action is described by a conjunctive precondition and postcondition over relevant properties of the system under attack. This is closely related to the syntax of classical planning formalisms. Furthermore, the attack graph is intended as an analysis of threats that arise through the possible *combinations* of these actions. This is, again, much as in classical planning. That said, attack graphs come in many different variants, and the term "attack graph" is rather overloaded. From our point of view here, relevant lines of distinction are the following.

In several early works (e. g. [31, 38]), the attack graph is the attack-action model itself, presented to the human as an abstracted overview of (atomic) threats. It was then proposed to instead reason about combinations of atomic threats, where the attack graph (also: "full" attack graph) is the state space arising from all possible sequencings of attack actions (e. g. [27, 32]). Later, positive formulations — positive preconditions and postconditions only — where suggested as a relevant special case, where attackers keep gaining new assets, but never lose any assets during the course of the attack [1, 10, 15, 24, 25, 38]. This restriction drastically simplifies the computational problem of non-probabilistic attack graph analysis, yet it also limits expressive power, especially in probabilistic models where a stochastic effect of an attack action (e. g., crashing a machine) may be detrimental to the attacker's objectives.

Probabilistic models of attack graphs/trees have been considered widely (e. g. [4, 5, 13, 20, 23, 30, 33]), and were later linked to to classical planning [2, 22]. More precise formulations in terms of *partially observable MDPs* were proposed for their ability to model incomplete knowledge on the attacker's side [29]. As POMDPs do not scale — neither in terms of modeling nor in terms of computation — it was thereafter proposed to use MDPs as a more scalable intermediate model [8, 12]. Here we build upon this latter model.

Stackelberg planning [34] models not only the attacker, but also the defender, and in that sense relates to more general game-theoretic security models. The most prominent application of such models thus far concerns physical infrastructures and defenses (e. g. [37]), quite different from the network security setting. A line of research considers attack-defense trees (e. g. [18, 19]), not based on standard sequential decision making formalisms. Some research considers pentesting but from an abstract theoretical perspective [3]. A basic difference to most game-theoretic models is that our mitigation analysis does not consider arbitrarily long exchanges of action and counter-action, but only a single such exchange: defender applies network fixes, attacker attacks the fixed network.

3 RUNNING EXAMPLE

We will use the following running example for easier introduction of our formalism and to foreshadow the modelling of networks which we will use in Section 5. Let us consider a network of five hosts, i.e., computers that are assigned an address at the network layer. It consists of a webserver W, an application server A, a database server D, and a workstation S. We partition the network into three zones called as follows: 1) the sensitive zone, which contains important assets, i.e., the database server D 2) the DMZ, which contains the services that need to be available from the outside, i.e., A and W, 3) the user zone, in which S is placed and 4) the internet, which is assumed under adversarial control by default and contains at least a host I.

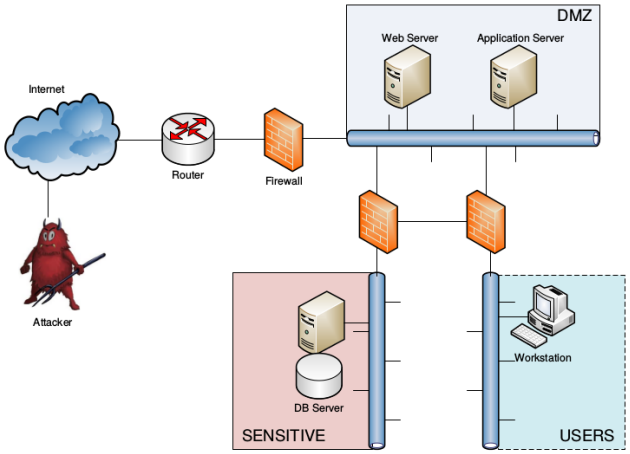


Figure 1: Network structure in our running example. (Figure adapted from Sarraute et al. [29].)

These zones are later (cf. Section 6) used to define the adversarial goals and may consist of several subnets. For now, each zone except the internet consists of exactly one subnet. These subnets are interconnected, with the exception of the internet, which is only connected to the DMZ. Firewalls filter some packets transmitted between the zones. We will assume that the webserver can be accessed via HTTPS (port 443) from the internet.

4 MITIGATION ANALYSIS AS STACKELBERG PLANNING

It was recently proposed to model penetration testing and mitigation tasks as Stackelberg planning task [34]. We review this formalism and show how vulnerability analysis can be mapped onto it.

Intuitively, the attacks we consider might make a service unavailable, but not physically remove a host from the network or add a physical connection between two hosts. We thus distinguish between network propositions and attacker propositions, where the former describes the network infrastructure and persistent configuration, while the latter describes the attacker's advance through the network. By means of this distinction, we may assume the state of the network to be fixed, while everything else can be manipulated by the attacker. The network state will, however, be altered during mitigation analysis, which we will discuss in more detail afterwards.

Networks are logically described through a finite set of *network propositions* P^{N}. A concrete *network state* is a subset of network propositions $s^{\mathsf{N}} \subseteq \mathsf{P}^{\mathsf{N}}$ that are true in this state. All propositions $p \notin s^{\mathsf{N}}$ are considered to be false.

Example 4.1. In the running example, the network topology is described in terms of network propositions $\mathsf{subnet}(s, h) \in \mathsf{P}^{\mathsf{N}}$ assigning a host h to a subnet s, e.g., $\mathsf{subnet}(\mathsf{sensitive}, D) \in \mathsf{P}^{\mathsf{N}}$. Connectivity is defined between subnets, e.g., $\mathsf{haclz}(\mathsf{internet}, \mathsf{dmz}, 443, \mathsf{tcp}) \in \mathsf{P}^{\mathsf{N}}$ indicates that TCP packets with destination port 443 (HTTPS) can pass from the internet into the DMZ. We assume that the webserver W, the workstation S and the database server D are vulnerable, e.g., $\mathsf{vul\_exists}(cve_W, W, 443, \mathsf{tcp}, \mathsf{integrity}) \in \mathsf{P}^{\mathsf{N}}$ for a vulnerability with CVE identifier cve_W affecting W on TCP port 443, that compromises integrity.

We formalize network penetration tests in terms of a probabilistic planning problem:

Definition 4.2 (penetration testing task [34]). A *penetration testing task* is a tuple $\Pi = (\mathsf{P}^{\mathsf{A}}, \mathsf{A}, \mathsf{I}^{\mathsf{A}}, \mathsf{G}, b_0^{\mathsf{A}})$ consisting of:

- a finite set of *attacker propositions* P^{A},
- a finite set of (probabilistic) *attacker actions* A (cf. Definition 4.4),
- the attacker's *initial state* $\mathsf{I}^{\mathsf{A}} \subseteq \mathsf{P}^{\mathsf{A}}$,
- a conjunction G over attacker proposition literals, called the *attacker goal*, and
- a non-negative attacker *budget* $b^{\mathsf{A}} \in \mathbb{R}^+ \cup \{\infty\}$, including the special case of an unlimited budget $b^{\mathsf{A}} = \infty$.

The objective in solving such a task — the attacker's objective — will be to maximize attack probability, i. e., to find action strategies maximizing the likelihood of reaching the goal, which we will specify in more detail. The attacker proposition are used to describe the state of the attack, e. g., dynamic aspects of the network and which hosts the attacker has gained access to.

Example 4.3. Consider an attacker that initially controls the internet, i.e., $\mathsf{controls}(I) \in \mathsf{I}^{\mathsf{A}}$ and has not yet caused W to crash, $\mathsf{available}(W) \in \mathsf{I}^{\mathsf{A}}$. The attacker's aim might be to inflict a privacy-loss on D, i.e., $\mathsf{compromised}(D, \mathsf{privacy})$, with a budget b^{A} of 3 units, which relate to the attacker actions below.

The attacks themselves are described in terms of actions which can depend on both network and attacker propositions, but only influence the attacker state.

Definition 4.4 (attacker actions [34]). An *attacker action* $a \in \mathsf{A}$ is a tuple $(pre^{\mathsf{N}}(a), pre^{\mathsf{A}}(a), c(a), O(a))$ where

- $pre^{\mathsf{N}}(a)$ is a conjunction over network proposition literals called the *network-state precondition*,
- $pre^{\mathsf{A}}(a)$ is a conjunction over attacker proposition literals called the *attacker-state precondition*,
- $c(a) \in \mathbb{R}^+$ is the *action cost*, and
- $O(a)$ is a finite set of *outcomes*, each $o \in O(a)$ consisting of an *outcome probability* $p(o) \in (0, 1]$ and a *postcondition* $post(o)$ over attacker proposition literals. We assume that $\sum_{o \in O(a)} p(o) = 1$.

The stochastic effect $post(o) \in O(a)$ can be used to model attacks that are probabilistic by nature, as well as to model incomplete knowledge (on the attacker's side) about the actual network configuration. Because $post(o)$ is limited to attacker propositions, we implicitly assume that the attacker cannot have a direct influence on the network itself. Although this is restrictive, it is a common assumption in the penetration testing literature (e. g. [10, 15, 24, 25]). The attacker action cost can be used to represent the effort the attacker has to put into executing what is being abstracted by the action. This can, e.g., be the estimated amount of time an action requires to be carried out, or the actual cost in terms of monetary expenses.

Example 4.5. If an attacker controls a host which can access a second host that runs a vulnerable service, it can compromise the second host w.r.t. privacy, integrity or availability, depending on the vulnerability. This is reflected, e.g., by an attacker action

$a \in \mathsf{A}$ which requires access to a vulnerable W within the DMZ, via the internet, s.t. $pre^{\mathsf{N}}(a) = \mathsf{subnet}(\mathsf{dmz}, W) \wedge \mathsf{subnet}(\mathsf{internet}, I) \wedge \mathsf{haclz}(\mathsf{internet}, \mathsf{dmz}, 443, \mathsf{tcp}) \wedge \mathsf{vul\_exists}(cve_W, W, 443, \mathsf{tcp}, \mathsf{integrity})$. In addition, I needs to be under adversarial control (which is the case initially), and W be available: $pre^{\mathsf{A}}(a) = \mathsf{controls}(I) \wedge \mathsf{available}(W)$.

The cost of this known vulnerability may be set to $c(a) = 1$, in which case the adversarial budget above relates to the number of such vulnerabilities used. More elaborate models are possible to distinguish known vulnerabilities from zero-day exploits which may exists, but only be bought or developed at high cost, or threats arising from social engineering.

We define three different outcomes $O(a) = \{o_{success}, o_{fail}, o_{crash}\}$ with probabilities

- $post(o_{success}) = \mathsf{compromised}(W, \mathsf{integrity}) \wedge \mathsf{controls}(W)$ in case the exploit succeeds,
- $post(o_{fail}) = \top$ in case the exploit has no effect and
- and $post(o_{crash}) = \neg\mathsf{available}(W)$ if it crashes W.

For example, we may have $p(o_{success}) = 0.5$, $p(o_{fail}) = 0.49$, and $p(o_{crash}) = 0.01$ because the exploit is of stochastic nature, with a small probability to crash the machine.

Regarding the first action outcome, $o_{success}$, note that we step here from a vulnerability that affects integrity, to the adversary gaining control over W. This is, of course, not a requirement of our formalism; it is a practical design decision that we make in our current model acquisition setup (and that was made by previous works on attack graphs with similar model acquisition machinery e. g. [25, 33]), because the vulnerability databases available do not distinguish between a privilege escalation and other forms of integrity violation. We get back to this in Section 5. Regarding the third action outcome, o_{crash}, note that negation is used to denote removal of literals, i. e., the following attacker state will not contain $\mathsf{available}(W)$ anymore, so that all vulnerabilities on W cease to be useful to the attacker.

The syntax and state transition semantics just specified is standard probabilistic planning. Thus, the state space of a penetration testing task can be viewed as a Markov decision process (MDP). A solution for an MDP is called policy and there are various objectives for these policies, i. e., notions of optimality, in the literature. For attack planning, arguably the most natural objective is *success probability*: the likelihood that the attack policy will reach a goal state.

Unfortunately, it is EXPTIME-complete to find such an optimal policy in general [21]. Furthermore, recent experiments have shown that, even with very specific restrictions on the action model, finding an optimal policy for a penetration testing task is feasible only for small networks of up to 25 hosts [36]. For the sake of scalability and following the lines of Stackelberg Planning [34], we thus focus on finding *critical attack paths*, instead of entire policies.[1] In a nutshell, a critical attack path is a sequence of actions whose success probability is maximal. We will also refer to such paths as *optimal attack plans*, or *optimal attack action sequences*. In contrast to policies, if any action within a critical attack path does not result in the desired outcome, we consider the attack to have failed. Critical attack paths are conservative approximations of optimal policies, i. e., the success probability of a critical attack path is a lower bound on the success probability of an optimal policy.

[1] Similar approximations have been made in the attack-graph literature. Huang et al. [14], e.g., try to identify critical parts of the attack-graph by analysing only a fraction thereof, in effect identifying only the most probable attacks.

Example 4.6. Reconsider the outcomes of action a from Example 4.5, $O(a) = \{o_{success}, o_{fail}, o_{crash}\}$. Assuming a reasonable set of attacker actions similar to the previous examples, no critical path will rely on the outcomes o_{fail} or o_{crash}, as otherwise a would be redundant or even counter-productive. Thus the distinction between these two kinds of failures becomes unnecessary, which is reflected in the models we generate in Section 5 and 6.

Finding possible attacks, e. g., through a penetration testing task as defined above, is only the first step in securing a network. Once these are identified, the analyst or the operator need to come up with a mitigation plan to mitigate or contain the identified weaknesses. This task can be formalized as follows.

Definition 4.7 (mitigation-analysis task [34]). Let P^{N} be a set of network propositions, and let $\Pi = (\mathsf{P}^{\mathsf{A}}, \mathsf{A}, \mathsf{I}^{\mathsf{A}}, \mathsf{G}, b_0^{\mathsf{A}})$ be a penetration testing task. A Π *mitigation-analysis task* is a triple $\mathsf{M} = (\mathsf{I}^{\mathsf{N}}, \mathsf{F}, b_0^{\mathsf{M}})$ consisting of

- the *initial network state* $\mathsf{I}^{\mathsf{N}} \subseteq \mathsf{P}^{\mathsf{N}}$,
- a finite set of *fix-actions* F, and
- the *mitigation budget* $b_0^{\mathsf{M}} \in \mathbb{R}^+ \cup \{\infty\}$.

The objective in solving such a task — the defender's objective —will be to find dominant mitigation strategies within the budget, i. e., fix-action sequences that reduce the attack probability as much as possible while spending the same cost. We now specify this in detail.

Fix-actions encode modifications of the network mitigating attacks simulated through Π.

Definition 4.8 (fix-actions [34]). Each fix-action $f \in \mathsf{F}$ is a triple $(pre(f), post(f), c^{\mathsf{M}}(f))$ of *precondition* $pre(f)$ and postcondition $post(f)$, both conjunctions over network proposition literals, and fix-action cost $c^{\mathsf{M}}(f) \in \mathbb{R}^+$.

We call f *applicable* to a network state s^{N} if $pre(f)$ is satisfied in s^{N}. The set of applicable f in s^{N} is denoted by $app(s^{\mathsf{N}})$. The result of this application is given by the state $s^{\mathsf{N}}[\![f]\!]$ which contains all propositions with positive occurrences in $post(f)$, and all propositions of s^{N} whose negation is not contained in $post(f)$.

Example 4.9. Removing a vulnerability by, e.g., applying a patch, is modelled as a fix-action f with $pre(f) = \mathsf{vul\_exists}(cve_W, W, 443, \mathsf{tcp}, \mathsf{integrity})$, $post(f) = \neg pre(f)$ and cost 1.

We can represent adding a firewall between the DMZ and the internet, assuming it was not present before, as a fix-action with $pre(f) = \mathsf{haclz}(\mathsf{internet}, \mathsf{dmz}, 443, \mathsf{tcp}) \wedge \neg\mathsf{fwapplied}(z_2)$, $post(f) = \neg\mathsf{haclz}(\mathsf{internet}, \mathsf{dmz}, 443, \mathsf{tcp}) \wedge \mathsf{fwapplied}(z_2)$ and cost 100. It is much cheaper to add a rule to an existing firewall than to add a firewall, which can be represented by a similar rule with $\mathsf{fwapplied}(z_2)$ instead of $\neg\mathsf{fwapplied}(z_2)$ in the precondition, and lower cost.

Note that, in contrast to attacker actions, fix-actions f are deterministic. A sequence of fix-actions can be applied to a network in order to lower the success probability of an attacker.

Definition 4.10 (mitigation strategy [34]). A sequence of fix-actions $\sigma = f_1, \ldots, f_n$ is called a *mitigation strategy* if it is applicable to the initial network state and its application cost is within the available mitigation budget.

To evaluate and compare different mitigation strategies, we consider their effect on the optimal attack. As discussed in the previous section, for the sake of scalability we use critical attack paths (optimal i. e. maximum-success-probability attack-action sequences) to gauge this effect, rather than full optimal MDP policies. As attacker actions in Π may contain a precondition on the network state, changing the network state affects the attacker actions in the state space of Π, and consequently the critical attack paths. To measure the impact of a mitigation strategy, we define $p^*(s^{\mathsf{N}})$ to be the success probability of a critical attack path in s^{N}, or $p^*(s^{\mathsf{N}}) = 0$ if there is no critical attack path (and thus there is no way in which the attacker can achieve its goal).

Definition 4.11 (dominance, solution [34]). Let σ_1, σ_2 be two mitigation strategies. σ_1 *dominates* σ_2 if

(i) $p^*(\mathsf{I}^{\mathsf{N}}[\![\sigma_1]\!]) < p^*(\mathsf{I}^{\mathsf{N}}[\![\sigma_2]\!])$ and $c^{\mathsf{M}}(\sigma_1) \leq c^{\mathsf{M}}(\sigma_2)$, or
(ii) $p^*(\mathsf{I}^{\mathsf{N}}[\![\sigma_1]\!]) \leq p^*(\mathsf{I}^{\mathsf{N}}[\![\sigma_2]\!])$ and $c^{\mathsf{M}}(\sigma_1) < c^{\mathsf{M}}(\sigma_2)$.

The *solution* $\mathcal{F}$ to M is the *Pareto frontier* of mitigation strategies σ: the set of σ that are not dominated by any other mitigation strategy.

5 PRACTICAL MODEL ACQUISITION

In this section, we describe a highly automated approach to acquire network models in practice, demonstrating our method to be readily applicable. Our workflow follows the same idea, but in addition we incorporate possible mitigation actions described in a concise and general schema. Moreover, our formalism considers the probabilistic/uncertain nature of exploits.

5.1 Workflow

This section describes the workflow for model acquisition and refinement via network scanning depicted in Figure 2. In the first step, the user scans a network using the Nessus tool, resulting in a report file. The user optionally describes the network topology in a JSON formatted topology file and sets the hosts that are initially assumed under adversarial control.[2] If this file is not given, we assume all hosts are interconnected w.r.t. every port that appears in the Nessus report. The user specifies the fixes the analysis should consider. Initially, this list is (automatically) populated by considering all known patches and a generic firewall rule that considers adding a firewall at all possible positions in the network, for the cost of five patches. The cost can be refined step by step, and patches that are not applicable, e.g., because of software incompatibilities, can be deleted from this file. The user can also refine the attacker budget and the mitigation budget. Initially, the attacker budget gives the number of exploits the attacker may use, as all exploits are assigned unit cost. With this information, the analysis gives a Pareto-optimal set of mitigation strategies within the given budget. After observing the fix-actions, the user may refine the fix-actions, as adopting some patches might be more expensive than others (which can be reflected in the associated mitigation costs), or some firewalls proposed might be too restrictive (which can be reflected by instantiating the firewall rule).

[2] In practice, penetration testers have access to firewall rules in machine-readable formats (e.g., Cisco, juniper), which can be used to create this file automatically.

5.2 Network Topology and Vulnerabilities

Like in Example 4.1, the network topology is given in terms of network predicates $\mathsf{subnet}(z, h) \in \mathsf{I}^{\mathsf{N}}$ for every host h in subnet z, $\mathsf{haclz}(z_1, z_2, \mathit{port}, \mathit{proto}) \in \mathsf{I}^{\mathsf{N}}$ for every z_1, from where all hosts in z_2 are reachable via $(\mathit{port}, \mathit{proto})$, which are derived from a JSON file, to allow for easy manual adjustment.

We translate the Nessus report to a set of network predicates $\mathsf{vul\_exists}(\mathit{cve}, h, \mathit{port}, \mathit{proto}, \mathit{type}) \in \mathsf{I}^{\mathsf{N}}$ for CVE *cve* affecting h on $(\mathit{port}, \mathit{proto})$, with effect on $\mathit{type} \in \{\mathsf{confidentiality}, \mathsf{integrity}, \mathsf{availability}\}$, and an attack-actions a for each z_1, h_1 in the universe of subnets and hosts, and $h_2 = h$, such that

$$\begin{aligned} \mathit{pre}^{\mathsf{N}}(a) = &\mathsf{subnet}(z_1, h_1) \wedge \mathsf{subnet}(z_2, h_2) \\ &\wedge \mathsf{haclz}(z_1, z_2, \mathit{port}, \mathit{proto}) \\ &\wedge \mathsf{vul\_exists}(\mathit{cve}, h_2, \mathit{port}, \mathit{proto}, \mathit{type}), \end{aligned}$$

and $O(a) = \{o_{\mathit{success}}, o_{\mathit{fail}}\}$. The value of *type* is determined from the U.S. government repository of standards based vulnerability management data, short NVD. As discussed in Example 4.6, the future availability of a host is disregarded by critical path analysis. Furthermore, the NVD does not provide data on potential side effects in case of failure. Thus, we assume all hosts in the network to be available throughout the attack.

We handle the success probability different from Example 4.5 by encoding it into the precondition, so an action with matching probability is chosen. More precisely, for all z_1, h_1 in the universe of subnets and hosts, and p' in the universe of probabilities, and $h_2 = h$, there is an action a with

$$\begin{aligned} \mathit{pre}^{\mathsf{N}}(a) = &\mathsf{subnet}(z_1, h_1) \wedge \mathsf{subnet}(z_2, h_2) \\ &\wedge \mathsf{haclz}(z_1, z_2, \mathit{port}, \mathit{proto}) \\ &\wedge \mathsf{vul\_exists}(\mathit{cve}, h_2, \mathit{port}, \mathit{proto}, \mathit{type}, p'), \end{aligned}$$

and $O(a) = \{o_{\mathit{success}}, o_{\mathit{fail}}\}$, with success probability $p(O_{\mathit{success}}) = p'$ and $p(O_{\mathit{fail}}) = 1 - p'$, $\mathit{post}(o_{\mathit{fail}}) = \top$. As a can only be applied if $p = p'$, this implies $p(o_{\mathit{success}}) = p$ for o_{success} the success outcome of a matching action. The matching action is uniquely determined, as in any reachable network state, there is at most one proposition $\mathsf{vul\_exists}(\mathit{cve}, h, \mathit{port}, \mathit{proto}, \mathit{type}, p)$ for any given *cve*, h, *port*, *proto* and *type*.

Today, the NVD does not provide data on how vulnerabilities may impact components other than the vulnerable component, e.g., in case of a privilege escalation. Such escalations are typically filed with $\mathit{type} = \mathsf{integrity}$. Hence we identify this vulnerability with a privilege escalation. Consequently, and as opposed to Example 4.5, $\mathit{pre}^{\mathsf{A}}(a) = \mathsf{compromised}(h, \mathsf{integrity})$, and $\mathit{post}(o_{\mathit{success}}) = \mathsf{compromised}(h, \mathit{type})$. CVSSv2 specifies one of three access vectors: 'local', which we ignore altogether, 'adjacent network', which models attacks that can only be mounted within the same subnet and typically pertain to the network layer, and 'network', which can be mounted from a different network. The second differs from the third in that the precondition requires z_1 and z_2 to be equal.

We assign probabilities according to the 'access complexity' metric, which combines the probability of finding an exploitable configuration, the probability of a probabilistic exploit to succeed, and the skill required to mount the attack into either 'low', 'medium' or 'high'. This is translated into a probability p of 0.2, 0.5, or 0.8, respectively. Thus $p(O_{\mathit{success}}) = p'$ and $p(O_{\mathit{fail}}) = 1 - p'$, where

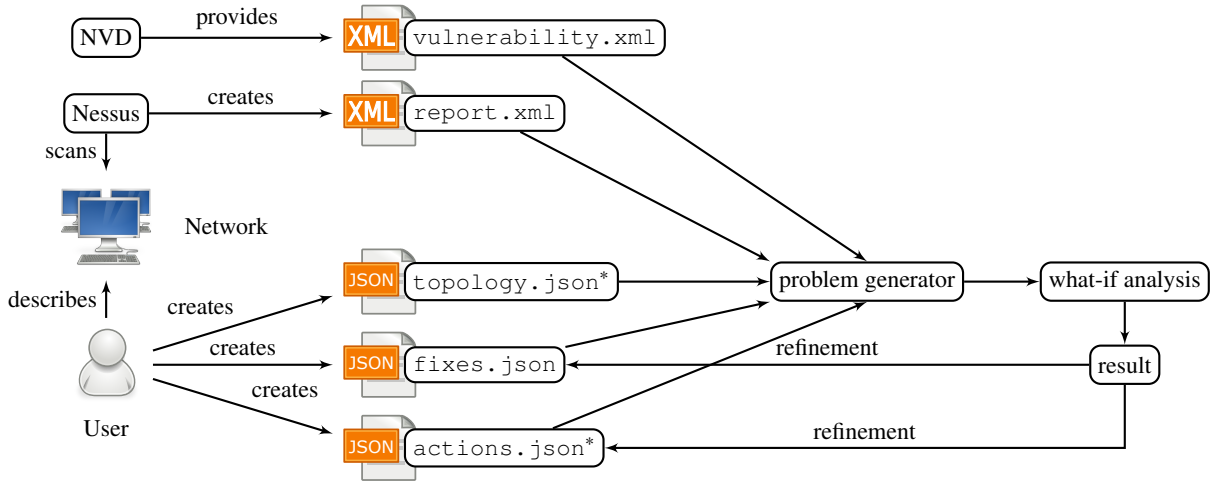


Figure 2: Workflow for model acquisition via network scanning, assuming a fixed attacker and mitigation budget. User input marked with * can be empty. The file `topology.json` can be left empty, in which case an open network is assumed.

$post(o_{fail}) = \top$. The action cost $c(a)$ is set to 1. A separate input file permits the user to refine both action cost and outcome probability of $o_{success}$ to reflect assumptions about the skill of the adversary and prior knowledge about the software configurations in the network.

5.3 Threat Model

The network configuration file defines subnets that are initially under attacker control, in which case compromised$(h, \text{integrity}) \in \mathsf{I}^{\mathsf{A}}$, and subnets which the attacker aims to compromise, in which case the goal condition is

$$\bigwedge_{(z,\,type)\text{ marked as target in }\texttt{topology.json}} \text{zcompromised}(z, type).$$

Additional artificial actions permit deriving zcompromised$(z, type)$ whenever compromised$(h, type) \wedge$ subnet(z, h).

5.4 Mitigation Model

Our formalism supports a wide range of fix-actions, but to facilitate its use, we provide three schemas, which we instantiate to a larger number of actions.

Fix schema. The fix schema models the application of existing patches, the development of missing patches and the implementation of local workarounds, e.g., application-level firewalls that protect systems from malicious traffic which are otherwise not fixable. The user specifies the CVE, host and port/protocol the fix applies to. Any of these may be a wild card *, in which case all matching fix actions of the form described in Example 4.9 are generated. The schema also includes the new probability assigned (which can be 0 to delete these actions) and an initial cost, which is applied the first time a fix-action instantiated from this schema is used, and normal cost which are applied for each subsequent use. Thus, the expensive development of a patch (high initial cost, low normal cost) can be compared with local workarounds that have higher marginal cost. The wild cards may be used to model available patches that apply to all hosts, as well as generic local workarounds that apply to any host, as a first approximation for the initial model.

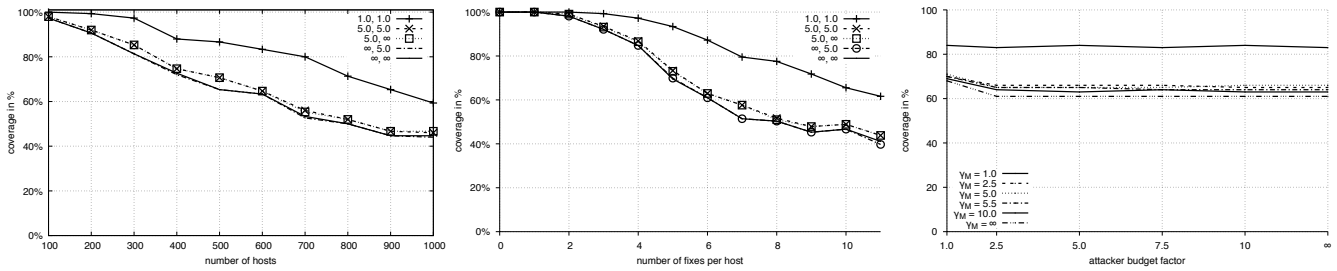
Non-zero probabilities may be used to model counter-measures which lower the success probability, but cannot remove it completely, e.g., address space layout randomisation. We employ a slightly indirect encoding to accommodate this case, adding additional attack-action copies for the lowered probability. The network state predicate determines uniquely which attack-action among these applies. The generated fix-action modifies the network state predicate accordingly.

Firewall schemata. There are two firewall schemas, one for firewalls between subnets, one for host-wise packet filtering. The former is defined by source and destination subnet along with port and protocol. Similar to the fix schema, any of the value may be specified, or left open as a wild card *, in which case a fix-action similar to the firewall fix in Example 4.9 is instantiated for every match. In addition, initial costs and cost for each subsequent application can be specified, in order to account for the fact that installing a firewall is more expensive than adding rules. The second firewall schema permits a similar treatment per host instead of subnets, which corresponds to local packet filtering rules.

6 EXPERIMENTS

It is easy to see that Stackelberg planning is PSPACE-hard. We hence explore the space of problems in which Stackelberg planning performs well enough to be useful. To provide an intuitive account of this space in terms of the network to be scanned, we we created a problem generator that produces network topologies and host configurations based on known vulnerabilities. This facilitates the performance evaluation of our mitigation analysis algorithm w.r.t. the number of hosts, fix actions and any combination of attacker and mitigation budget. For details to the generator, we refer the reader to the long version of this paper [35].

We evaluate our model using Speicher et. al's Stackelberg planning algorithm [34] which was implemented on top of the FD planning tool [11]. Our experiments were conducted on a cluster of Intel Xeon E5-2660 machines running at 2.20 GHz. We terminated a run if the Pareto frontier was not found within 30 minutes, or the process required more than 4 GB of memory during execution.



(a) (b) (c)

Figure 3: % instances solved within time (memory) limits. (a) for scaling number of hosts and 5 fix-actions per host on average, (b) for scaling number of fix-actions per hosts, but fixing $H = 500$, and (c) scaling budgets, but fixing the number of hosts to 500 and fixing the number of fixes to 5 per host.

In our evaluation we focus on coverage values, i.e. the number of instances that could be solved within the time (memory) limits. We investigate how coverage is affected by (1) scaling the network size, (2) scaling the number of fix actions, and (3) the mitigation budget, respectively the attacker budget.

The budgets are computed as follows. In a precomputation step, we compute the minimal attacker budget b^{A}_{min} that is required for non-zero success probability $p^*(\mathsf{I}^{\mathsf{N}})$. The minimal mitigation budget b^{M}_{min} is then set to the minimal budget required to lower the attacker success probability with initial attacker budget $b^{\mathsf{A}}_0 = b^{\mathsf{A}}_{min}$. We experimented with budget values relative to those minimal budget values, resulting from scaling them by factors out of $\{1, 2.5, 5, 7.5, 10, \infty\}$. We denote γ_{M} the factor which is used to scale the mitigation budget, and vice versus γ_{A} the factor for the attacker budget.

In Figure 3(a), we observe that the algorithm provides reasonable coverage > 50% for up to 800 hosts, when considering on average 5 vulnerabilities and 5 fix-actions per host. Unless both the attacker and the mitigation budget are scaled to 1 (relative to γ_{M} and γ_{A}, respectively), this result is relatively independent from the budget. One explanation why it is independent from the budget is that there is no huge difference between factors 5 and ∞ in the sense that the attacker cannot find more or better critical paths and the defender cannot find more interesting fix action sequences because of the infinite budgets. In the case that both are scaled to 1, the searches for critical paths and fix actions sequences are vastly simplified. Hence the overall coverage is better. Note that the number of fix actions scales linearly with the number of hosts, which in the worst case, i.e., when all sequences need to be regarded, leads to an exponential blowup.

In Figure 3(b), we have fixed the number of hosts to 500, but varied the number of fixes that apply per host by scaling λ_F in integer steps from 0 to 10, which controls the expected value of patch fixes generated per host. We then plotted the coverage with the total number of fixes, i.e., the number of firewall fixes and patch fixes actually generated. We tested 50 samples per value of λ_F and attacker/mitigation budget. We cut of at above 11 fixes per host, where we had too few data points. We furthermore applied a sliding average with a window size of 1 to smoothen the results, as the total number of actual fixes varies for a given λ_F. Similar to Figure 3(a), the influence of the attacker and mitigation budget is less than expected, except for the extreme case where both are set to their minimal values. The results suggest that the mitigation analysis is reliable up to a number of 4 fixes per hosts, but up to 16 fixes per host, there is still a decent chance for termination.

Figure 3(c) compares the impact of the mitigation- and attacker-budget factors $\gamma_{\mathsf{M}}, \gamma_{\mathsf{A}} \in \{1, 2.5, 5, 7.5, 10, \infty\}$. The overall picture supports our previous observations. The attacker budget has almost no influence on the performance of the algorithm. This, however, is somewhat surprising given that the attacker budget not only affects the penetration testing task itself, but also influences the mitigation-analysis. Larger attacker budgets in principle allow for more attacks, imposing the requirement to consider more expensive mitigation strategies. It will be interesting to explore this effect, or lack thereof, on real-life networks.

In contrast, the algorithm behaves much more sensitive to changes in the mitigation budget. Especially in the step fom $\gamma_{\mathsf{M}} = 1.0$ to $\gamma_{\mathsf{M}} = 2.5$, coverage decreases significantly (almost 20 percentage points regardless of the attacker budget value). This can be explained by the effect of the increased mitigation budget on the search space. However, further increasing the mitigation budget has a less severe effect.

7 CONCLUSION & FUTURE WORK

The mitigation analysis method presented in this work is the first of its kind and provides a semantically clear and thorough methodology for analysing mitigation strategies. We leverage the fact that network attackers can be simulated, and hence strategies for mitigation can be compared before being implemented. We have presented a highly automated modelling approach along with an iterative workflow. Based on a detailed network and configuration model, we demonstrated the feasibility of the approach and scalability of the algorithm.

Two major ongoing and future lines of work arise from this contribution, pertaining to more effective algorithms, and to the practical acquisition of more refined models. Regarding effective algorithms, the effective computation of the Pareto frontier stands and falls with the speed with which a first good solution — a cheap fix-action sequence reducing attacker success probability to a small value — is found. Finding good solutions quickly is precisely the mission statement of heuristic functions in AI heuristic search procedures, which typically operate by solving a *relaxed* (simplified) version of the problem to deliver lower bounds. The key difficulty is the move-countermove pattern in Stackelberg planning, which requires a new understanding of what a relaxation ought to achieve in this setting.

Regarding the model acquisition of more refined models, there is a trade-off between the accuracy of the model, and the degree of automation vs. manual effort with which the model is created. First, economically, a more detailed machine-readable description of vulnerabilities cost money, hence there needs to be an incentive to provide this data. The successful commercial use of simulated pentesting at Core Security shows that there is money to be made with fine-grained vulnerability data. We hope that mitigation analysis methods such as ours will be adopted and provide further incentives, as centralised knowledge about the nature of vulnerabilities can be used to improve analysis and hence lower mitigation cost. Declarative descriptions like OVAL are well-suited to this end.

Second, conceptually, the transitivity in network attacks is not understood well enough. Due to the lack of additional information, we assume that integrity violations allow for full host compromise, which is an over-approximation. While CVSSv3 provides a metric distinguishing attacks that switch scope, it is unclear how exactly this could be of use, as the scope might pertain to user privileges within a service, sandboxes, system users, dom0-privileges etc. A formal model for privilege escalation could be used to describe the effect if a vulnerability in an abstract manner that can be instantiated into a concrete outcome once an actual software configuration is given and form the basis for the automated acquisition of realistic network models.

Acknowledgments. This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA, grant no. 16KIS0656).

REFERENCES

[1] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. 2002. Scalable, graph-based network vulnerability analysis. In *ACM Conference on Computer and Communications Security*. 217–224.

[2] Mark Boddy, Jonathan Gohde, Tom Haigh, and Steven Harp. 2005. Course of Action Generation for Cyber Security Using Classical Planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, Susanne Biundo, Karen Myers, and Kanna Rajan (Eds.). Morgan Kaufmann, Monterey, CA, USA, 12–21.

[3] Rainer Böhme and Márk Félegyházi. 2010. Optimal Information Security Investment with Penetration Testing. In *Proceedings of the 1st International Conference on Decision and Game Theory for Security (GameSec'10)*. 21–37.

[4] Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson. 2006. Rational Choice of Security Measures Via Multi-parameter Attack Trees. In *1st International Workshop on Critical Information Infrastructures Security (CRITIS'06)*. 235–248.

[5] Ahto Buldas and Roman Stepanenko. 2012. Upper Bounds for Adversaries' Utility in Attack Trees. In *Proceedings of the 3rd International Conference on Decision and Game Theory for Security (GameSec'12)*. 98–117.

[6] Alan Calder and Geraint Williams. 2014. *PCI DSS: A Pocket Guide, 3rd Edition*. IT Governance Publishing.

[7] Core Security SDI Corporation. [n. d.]. Core IMPACT. https://www.coresecurity.com/core-impact, Core IMPACT uses model-based attack planning since 2010).

[8] Karel Durkota and Viliam Lisý. 2014. Computing Optimal Policies for Attack Graphs with Action Failures and Costs. In *7th European Starting AI Researcher Symposium (STAIRS'14)*.

[9] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

[10] Nirnay Ghosh and S. K. Ghosh. 2009. An Intelligent Technique for Generating Minimal Attack Graph. In *Proceedings of the 1st Workshop on Intelligent Security (SecArt'09)*.

[11] Malte Helmert. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26 (2006), 191–246.

[12] Jörg Hoffmann. 2015. Simulated Penetration Testing: From "Dijkstra" to "Turing Test++". In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein (Eds.). AAAI Press.

[13] John Homer, Su Zhang, Xinming Ou, David Schmidt, Yanhui Du, S. Raj Rajagopalan, and Anoop Singhal. 2013. Aggregating vulnerability metrics in enterprise networks using attack graphs. *Journal of Computer Security* 21, 4 (2013), 561–597.

[14] Heqing Huang, Su Zhang, Xinming Ou, Atul Prakash, and Karem A. Sakallah. 2011. Distilling critical attack graph surface iteratively through minimum-cost SAT solving. In *27th Annual Computer Security Applications Conference (ACSAC)*. 31–40.

[15] Sushil Jajodia, Steven Noel, and Brian O'Berry. 2005. Topological Analysis of Network Attack Vulnerability. In *Managing Cyber Threats: Issues, Approaches and Challenges*. Chapter 5.

[16] Alexander Koller and Jörg Hoffmann. 2010. Waking Up a Sleeping Rabbit: On Natural-Language Sentence Generation with FF. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*. AAAI Press.

[17] Alexander Koller and Ronald Petrick. 2011. Experiences with Planning for Natural Language Generation. *Computational Intelligence* 27, 1 (2011), 23–40.

[18] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. 2013. ADTool: Security Analysis with Attack-Defense Trees. In *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST'13)*. 173–176.

[19] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. 2010. Foundations of Attack-Defense Trees. In *Proceedings of the 7th International Workshop on Formal Aspects in Security and Trust (FAST'10)*. 80–95.

[20] Viliam Lisý and Radek Píbil. 2013. Computing Optimal Attack Strategies Using Unconstrained Influence Diagrams. In *Pacific Asia Workshop on Intelligence and Security Informatics*. 38–46.

[21] Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. 1998. The Computational Complexity of Probabilistic Planning. *Journal of Artificial Intelligence Research* 9 (1998), 1–36. http://jair.org/abstracts/littman98a.html

[22] Jorge Lucangeli, Carlos Sarraute, and Gerardo Richarte. 2010. Attack Planning in the Real World. In *Proceedings of the 2nd Workshop on Intelligent Security (SecArt'10)*.

[23] Margus Niitsoo. 2010. Optimal Adversary Behavior for the Serial Model of Financial Attack Trees. In *Proceedings of the 5th International Conference on Advances in Information and Computer Security (IWSEC'10)*. 354–370.

[24] Steven Noel, Matthew Elder, Sushil Jajodia, Pramod Kalapa, Scott O'Hare, and Kenneth Prole. 2009. Advances in Topological Vulnerability Analysis. In *Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security (CATCH'09)*. 124–129.

[25] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. 2006. A scalable approach to attack graph generation. In *ACM Conference on Computer and Communications Security*. 336–345.

[26] Cynthia Phillips and Laura Painton Swiler. 1998. A Graph-Based System for Network-Vulnerability Analysis. In *Proceedings of the New Security Paradigms Workshop*.

[27] Ronald W. Ritchey and Paul Ammann. 2000. Using Model Checking to Analyze Network Vulnerabilities. In *IEEE Symposium on Security and Privacy*. 156–165.

[28] Wheeler Ruml, Minh Binh Do, Rong Zhou, and Markus P. J. Fromherz. 2011. On-line Planning and Scheduling: An Application to Controlling Modular Printers. *Journal of Artificial Intelligence Research* 40 (2011), 415–468.

[29] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. 2012. POMDPs Make Better Hackers: Accounting for Uncertainty in Penetration Testing. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, Jörg Hoffmann and Bart Selman (Eds.). AAAI Press, Toronto, ON, Canada, 1816–1824.

[30] Carlos Sarraute, Gerardo Richarte, and Jorge Lucángeli Obes. 2011. An algorithm to find optimal attack paths in nondeterministic scenarios. In *Workshop on Security and Artificial Intelligence*. 71–80.

[31] B. Schneier. 1999. Attack Trees. *Dr. Dobbs Journal* (1999).

[32] Oleg Sheyner, Joshua W. Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. 2002. Automated Generation and Analysis of Attack Graphs. In *IEEE Symposium on Security and Privacy*. 273–284.

[33] Anoop Singhal and Xinming Ou. 2011. *Security risk analysis of enterprise networks using probabilistic attack graphs*. Technical Report. NIST Interagency Report 7788.

[34] Patrick Speicher, Marcel Steinmetz, Michael Backes, Jörg Hoffmann, and Robert Künnemann. 2018. Stackelberg Planning: Towards Effective Leader-Follower State Space Search. In *AAAI'18*.

[35] Patrick Speicher, Marcel Steinmetz, Jörg Hoffmann, Michael Backes, and Robert Künnemann. 2018. *Towards Automated Network Mitigation Analysis (extended)*. Technical Report. arXiv:1705.05088v2 http://arxiv.org/abs/1705.05088v2

[36] Marcel Steinmetz, Jörg Hoffmann, and Olivier Buffet. 2016. Goal Probability Analysis in MDP Probabilistic Planning: Exploring and Enhancing the State of the Art. *Journal of Artificial Intelligence Research* 57 (2016), 229–271.

[37] Milind Tambe. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

[38] Steven J. Templeton and Karl E. Levitt. 2000. A requires/provides model for computer attacks. In *Proceedings of the Workshop on New Security Paradigms (NSPW'00)*. 31–38.