Simulated Penetration Testing and Mitigation Analysis

Michael Backes
CISPA, Saarland University & MPI-SWS
Saarland Informatics Campus
backes@mpi-sws.org

Jörg Hoffmann
CISPA, Saarland University
Saarland Informatics Campus
hoffmann@cs.uni-saarland.de

Robert Künnemann
CISPA, Saarland University
Saarland Informatics Campus
robert.kuennemann@cispa.saarland

Patrick Speicher
CISPA, Saarland University
Saarland Informatics Campus
patrick.speicher@cispa.saarland

Marcel Steinmetz
CISPA, Saarland University
Saarland Informatics Campus
steinmetz@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. 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 what-if analysis 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.

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' [10]. Network penetration tests 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 penetration test: 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 [4], [33] and in fact used commercially, at a company called Core Security, since at least 2010 [11]. These approaches, which derive from earlier approaches based on attack graphs [40], [45], [46], 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 [12], [19] (we discuss this in detail as part of our related work discussion, Section 2).

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

The algorithm we propose optimizes the mitigation strategy based on a set of possible mitigation actions. 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. While, e.g., an application-level firewall might be an efficient temporary workaround for a vulnerability that affects a single host, contracting a software vendor to provide a patch might be more cost-efficient in case the vulnerability appears throughout the network. To reflect cases like this, mitigation actions are assigned a cost for their first application (set-up cost), and another potentially different cost for all subsequent applications (application cost). 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. This min-max notion is similar to a Stackelberg game, which are frequently used in security games [29]. The foundational assumption is that the defender acts first, while the adversary can chose her best response after observing this choice, similar to a market leader and her followers. The algorithm thus provides a well-founded basis for a holistic mitigation strategy.

After discussing related work in Section 2 and giving a running example in Section 3, we present our mitigation analysis models and algorithms in Sections 4 to 6, framed in a formalism suited for a large range of mitigation/attack planning problems. In Section 7, we show that a particular class of these models can be derived 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 8, 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 [13] 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. In *partially observable probabilistic planning*, we are in addition given a probability distribution over the possible initial states, so the underlying state model is a partially observable MDP (POMDP).

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, [42], natural language sentence generation [25], [26], greenhouse logistics [18], and, in particular, network security penetration testing [4], [12], [19], [33], [43]. This latter branch of research – network attack planning as a tool for automated security testing – has been coined simulated pentesting, and is what we continue here.

Simulated pentesting is rooted in the consideration of attack graphs, first introduced by Philipps and Swiler [40]. 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 distinction lines are the following.

In several early works (e. g. [45], [50]), 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. [41], [46]). 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 [2], [14], [23], [36], [37], [50]. 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.[1]

A close relative of attack graphs are *attack trees* (e. g. [34], [45]). These arose from early attack graph variants, and developed into "Graphical Security Models" [27]: Directed acyclic AND/OR graphs organizing known possible attacks into a top-down refinement hierarchy. The human user writes that hierarchy, and the computer analyzes how attack costs and probabilities propagate through the hierarchy. In comparison to attack graphs and planning formulations, this has computational advantages, but cannot find unexpected attacks, arising from unforeseen combinations of atomic actions.

Probabilistic models of attack graphs/trees have been considered widely (e. g. [8], [9], [21], [30], [35], [44], [47]), yet they weren't, at first, given a formal semantics in terms of standard sequential decision making formalisms. The latter was done later on by the AI community in the simulated pentesting branch of research. After initial works linking non-probabilistic attack graphs to classical planning [4], [33], Sarraute et al. [43] devised a comprehensive model based on POMDPs, designed to capture penetration testing as precisely as possible, explicitly modeling the incomplete knowledge on the attacker's side, as well as the development of that knowledge during the attack. 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 [12], [19]. Here we build upon this latter model, extending it with automated support for mitigation analysis.

Mitigation analysis models not only the attacker, but also the defender, and in that sense relates to game-theoretic security models. The most prominent application of such models thus far concerns physical infrastructures and defenses (e. g. [49]), quite different from the network security setting. A line of research considers attack-defense trees (e. g. [27], [28], not based on standard sequential decision making formalisms. Some research considers pentesting but from a very abstract theoretical perspective [5]. 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. The latter relates to Stackelberg competitions, yet with interacting state-space search models underlying each side of the game.

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 7. 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 and the information it stores, 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.

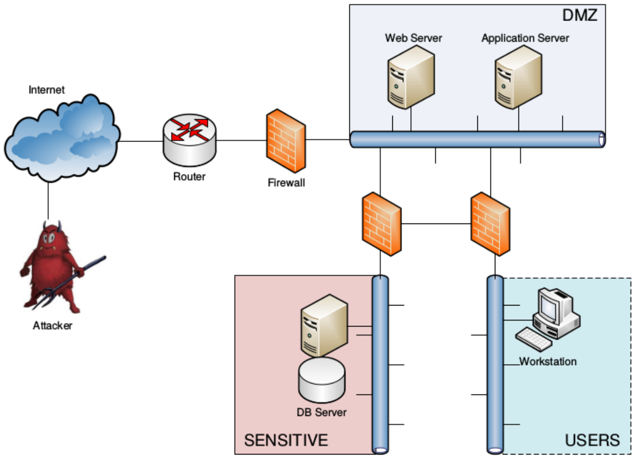


Figure 1. Network structure in our running example. (Figure adapted from Sarraute et al. [43].)

These zones are later (cf. Section 8) 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. Penetration Testing Formalism

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 in Section 5.

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.

[1] The restriction to positive preconditions and postconditions is actually known in Automated Planning not as a planning problem of interest in its own right, but as a problem *relaxation*, serving for the estimation of goal distance to guide search on the actual problem [6], [20].

Example 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 1 (penetration testing task) *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 2),*
- *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. We now specify this in 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 2 *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 2 (attacker actions) *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 network-state precondition $pre^\mathsf{N}(a)$, attacker-state precondition $pre^\mathsf{A}(a)$ and postconditions $post(o)$ represent the conditions under which a can be applied as well as the stochastic effect of the application of a: $post(o) \in O(a)$ holds after the application of a with probability $p(o)$. This 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 very restrictive, it is a common assumption in the penetration testing literature (e. g.. [14], [23], [36], [37]). 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 for example be the estimated amount of time an action requires to be carried out, or the actual cost in terms of monetary expenses.

Example 3 *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.*

$$\begin{aligned} pre^\mathsf{N}(a) =& \mathsf{subnet}(\mathsf{dmz}, W) \wedge \mathsf{subnet}(\mathsf{internet}, I) \\ & \wedge \mathsf{haclz}(\mathsf{internet}, \mathsf{dmz}, 443, \mathsf{tcp}) \\ & \wedge \mathsf{vul\_exists}(cve_W, W, 443, \mathsf{tcp}, \mathsf{integrity}) \end{aligned}$$

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 maybe 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.*

There could be three different outcomes $O(a) = \{o_{success}, o_{fail}, o_{crash}\}$*, with different 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* $post(o_{crash}) = \neg\mathsf{available}(W)$ *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 (but not negligible) 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. [37], [47]), 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 7.*

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.*

Assume a fixed penetration testing task. Given some network state, we can now define the *state space*, in which attacks are computed.

Definition 3 (state space) *The* state space *of Π in the network state s^{N} is the probabilistic transition system* $(\mathsf{S}, \mathsf{T}, s_0, \mathsf{S}_\top)$ *where*

- S *is the set of* attacker states*, or* states *for short. Each state $s \in \mathsf{S}$ is associated with the set of attacker propositions $\mathsf{P}^{\mathsf{A}}(s) \subseteq \mathsf{P}^{\mathsf{A}}$ true in s, and the* remaining budget $b^{\mathsf{A}}(s) \in \mathbb{R}_0^+ \cup \{\infty\}$.
- $\mathsf{T} : \mathsf{S} \times \mathsf{A} \times \mathsf{S} \mapsto [0, 1]$ *is the* transition probability function*, and corresponds to the application of attacker actions to states. An attacker action $a \in \mathsf{A}$ is* applicable *to a state $s \in \mathsf{S}$ in s^{N} if the network precondition $pre^{\mathsf{N}}(a)$ is satisfied in s^{N}, the attacker precondition $pre^{\mathsf{A}}(a)$ is satisfied in $\mathsf{P}^{\mathsf{A}}(s)$, and there is enough budget in s for the application of a, i. e., $b^{\mathsf{A}}(s) \geq c(a)$. The result of an outcome $o \in O(a)$ in s is the state $s[\![o]\!]$ where $\mathsf{P}^{\mathsf{A}}(s[\![o]\!])$ contains all propositions $p \in \mathsf{P}^{\mathsf{A}}$ that are contained in $post(o)$ and all propositions $p \in \mathsf{P}^{\mathsf{A}}(s)$ whose negation does not occur in $post(o)$. We define $b^{\mathsf{A}}(s[\![o]\!]) = b^{\mathsf{A}}(s) - c(a)$. For states $s, t \in \mathsf{S}$ and action $a \in \mathsf{A}$, the transition probabilities are then defined as $T(s, a, t) = p(o)$ if a is applicable to s and $t = s[\![o]\!]$ for $o \in O(a)$,*[2] *and $T(s, a, t) = 0$ otherwise.*
- s_0 *is the* initial state *where $\mathsf{P}^{\mathsf{A}}(s_0) = \mathsf{I}^{\mathsf{A}}$ and $b^{\mathsf{A}}(s_0) = b_0^{\mathsf{A}}$.*
- $\mathsf{S}_\top \subseteq \mathsf{S}$ *is the set of* goal states*, where $s \in \mathsf{S}_\top$ if G is satisfied in $\mathsf{P}^{\mathsf{A}}(s)$.*

Viewing the state space of Π as a Markov decision process (MDP), an *attack* in Π for the network state s^{N} is a solution to that MDP, i. e., a *policy*. A policy is a partial function $\pi : \mathsf{S} \mapsto \mathsf{A}$ where (1) for every $s \in \mathsf{S}$ where $\pi(s)$ is defined, $\pi(s)$ is applicable to s in s^{N}; and (2) π is closed, i. e., π is defined for every s reachable under π from the initial state s_0.

There are various objectives for MDP 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, finding such an optimal policy is EXPTIME-complete in general [32]. 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 [48]. For the sake of scalability we thus focus on finding *critical attack paths*, instead of entire policies.[3]

Definition 4 (critical attack path) *A* critical attack path *in the network state s^{N} is a path $s_0, a_1, s_1, \ldots, a_n, s_n$ within the state space of Π in s^{N}, that starts in an initial state s_0, ends in a goal state $s_n \in \mathsf{S}_\top$, and maximizes $\prod_{i=1}^{n} T(s_{i-1}, a_i, s_i)$ among all paths from s_0 to any goal state.*

In other words, 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.

Example 4 *Reconsider the outcomes of action a from Example 3, $O(a) = \{o_{success}, o_{fail}, o_{crash}\}$. 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 counterproductive. Thus the distinction between these two kinds of failures becomes unnecessary, which is reflected in the models we generate in Section 7 and 8. The downside of considering only single paths instead of policies can be observed in the following example. Consider the case where a second action a' has similar outcomes $O(a') = \{o'_{success}, o'_{fail}, o'_{crash}\}$ to a, but $p(o'_{success}) < p(o_{success})$ while $p(o'_{crash})$ is considerably smaller than $p(o_{crash})$. Assuming that W is the only host that can be used to reach S or D, an optimal policy might chose a' in favour of a, while a critical attack path will insist on a.*

5. Mitigation Analysis Formalism

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 5 (mitigation-analysis task) *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 $b_0^{\mathsf{M}} \in \mathbb{R}^+ \cup \{\infty\}$.

[2] We assume here that each outcome $o \in O(a)$ leads to a different state.

[3] Similar approximations have been made in the attack-graph literature. Huang et al. [22], 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.

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 6 (fix-actions) *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 5 *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.*

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 may represent adding a firewall between the DMZ and the internet (assuming it was not present before, i.e., $\mathsf{fwapplied}(z_2) \notin \mathsf{I}^{\mathsf{N}}$). 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 7 (mitigation strategy) *A sequence of fix-actions $\sigma = f_1, \dots, f_n$ is called a* mitigation strategy *if it is applicable to the initial network state and its application cost is within the available mitigation budget, where*

- *$f_1, \dots, f_n$ are said to be applicable to a network state s^{N} if f_1 is applicable to s^{N} and $f_2, \dots, f_n$ are applicable to $s^{\mathsf{N}}[\![f_1]\!]$. The resulting state is denoted $s^{\mathsf{N}}[\![f_1, \dots, f_n]\!]$.*
- *The application cost of $f_1, \dots, f_n$ is $c^{\mathsf{M}}(f_1, \dots, f_n) = \sum_{i=1}^{n} c^{\mathsf{M}}(f_i)$.*

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 8 (dominance, solution) *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.*

In other words, we consider a mitigation strategy σ_1 *better* than another one, σ_2, if either σ_1 reduces the probability of an successful attack to the network more, while not imposing a higher cost, or σ_1 costs less than σ_2 while it lowers the success probability of an attack at least by the same amount. The solution to our mitigation-analysis task is the set of *dominant* (non-dominated) mitigation strategies.

This is similar Stackelberg games, in which a market leader moves first and is followed by one or more market followers, and thus optimises his strategy w.r.t. their best response. Stackelberg games in the two-player setting are frequently used in security settings [29].

It is easy to see that our notion of solutions is well-defined, in the following way:

Theorem 1 *The solution $\mathcal{F}$ to M always exists, is non-empty, is unique, and is finite.*

Proof: As we assumed that all fix-actions f have positive cost, it immediately follows that the empty mitigation strategy $\sigma = \epsilon$ is not dominated by any other mitigation strategy, and hence $\sigma \in \mathcal{F}$. $\mathcal{F}$ is unique since it must contain all non-dominated mitigation strategies. Coming to the last part, assume for contradiction that $\mathcal{F}$ is not finite. As the number of different network propositions is finite, the number of different network states is finite as well. Therefore, there must be a network state s^{N} that is reached from the initial network state by infinitely many mitigation strategies in $\mathcal{F}$. As all fix-actions have positive cost, there must in particular be two mitigation strategies $\sigma_1, \sigma_2 \in \mathcal{F}$ so that $\mathsf{I}^{\mathsf{N}}[\![\sigma_1]\!] = \mathsf{I}^{\mathsf{N}}[\![\sigma_2]\!]$ and $c^{\mathsf{M}}(\sigma_1) < c^{\mathsf{M}}(\sigma_2)$, i. e., so that σ_1 dominates σ_2, in contradiction to the definition of $\mathcal{F}$. ■

6. Analysis Algorithms

We actually want to compute $\mathcal{F}$ with reasonable efficiency. We thus specify how we compute critical paths and solve mitigation tasks as a whole.

6.1. Penetration Testing

We compute critical attack paths through a compilation from network penetration testing tasks to classical, deterministic, planning formalisms. The latter can then be solved using standard algorithms for finding minimal-cost plans.

This compilation is compromised of two steps. First, in order to get rid of stochastic action outcomes, we apply the *all-outcome determinization* (e. g. [31], [53]). That is, we create a deterministic action a^o for every attacker action a and stochastic outcome $o \in O(a)$, where a^o has the same precondition than a, and the postcondition of o. Second, to get classical planning methods output attack sequences with highest chances of success, instead of minimal cost, we encode the outcome probability $p(o)$ as action costs: $c(a^o) = -\log(p(o))$ (cf. [24]). The attack $a_1^o, \ldots, a_n^o$ resulting from the classical planning method is guaranteed to have minimal cost $\sum_{i=1}^n c(a_i^o)$, and hence $\prod_{i=1}^n p(o_i)$ must be maximal, i. e., $a_1, \ldots, a_n$ is a critical attack path.

Given this encoding, the attack-planning problem can be solved with standard planning algorithms and tools. The state of the art consists in *heuristic search* methods [39], which employ search guidance through *heuristic functions* – functions mapping states to estimates of cost-to-goal – to find optimal solutions quickly. In our implementation, we use an extension of the FD system [16], with the LM-cut heuristic function [17].

6.2. Mitigation Analysis

In this section, we formally define the mitigation analysis algorithm used and the pruning techniques employed to improve performance. Finally, we show this techniques correct.

We compute the solution to a given mitigation-analysis task M, i.e., the Pareto frontier w.r.t. Definition 8, using a depth-oriented tree search algorithm. While a naïve implementation needs to consider every sequence of fix actions over F for inclusion in the global Pareto frontier $\mathcal{F}$, often enough it is sufficient to consider subsets of F, as most fix-actions are commutative and thus the analysis invariant w.r.t. permutations. This is particularly relevant for attack mitigations, as fixes are often local and independent, however, commutativity is not always given, consider, e.g., the firewall rule discussed in Example 5. As a firewall needs to be acquired before firewall rules can be added cheaply, this imposes a constraint that we formalize in the notion of commutativity. We define commutativity on top of interference [52] which we will also need later on.

Definition 9 (interference, commutativity) *Let Π be a mitigation-analysis task with network propositions P^N and fix-actions F, and let $f_1, f_2 \in \mathsf{F}$.*

1. *Action f_1* disables *f_2 if there exists a proposition literal $p \in post(f_1)$ and $\neg p \in pre(f_2)$ or vice versa.*
2. *Actions f_1 and f_2* conflict *if there exists a proposition literal p such that $p \in post(f_1)$ and $\neg p \in post(f_2)$ or vice versa.*
3. *Actions f_1 and f_2* interfere *if they conflict or either disable the other. We write $\inf(f)$ for the set of actions $f' \in \mathsf{F}$ with which f interferes.*
4. *Action f_1* enables *f_2 if there exists a proposition $p \in post(f_1)$ and $p \in pre(f_2)$ or vice versa.*
5. *Actions f_1 and f_2 are* commutative *if they do not* interfere *and not enable the other.*

The interference and commutativity relations on elements from F can both be computed up front. To avoid considering permutations of commutative actions, we apply a transition reduction technique based on so-called *sleep sets* [15], [52]. A sleep set for a sequence σ is a set of operators f that are applicable after σ but skipped during search. When expanding successor actions for σ, we only consider applicable actions outside $\mathit{sleep}(\sigma)$. Let $f_1, ..., f_n$ be these actions, ordered in the same way as they are considered by the search algorithm. For each successor path $\sigma \circ \langle f \rangle$, we set $\mathit{sleep}(\sigma \circ \langle f \rangle) := (\mathit{sleep}(\sigma) \cup \{f_1, ..., f_{i-1}\}) \setminus \{f \mid f, f_i \text{ are not commutative}\}$.

We globally maintain *a*) C^0, the current bound for the cost of lowering the attacker probability to zero, in order to prune sequences that are dominated from the start, *b*) Ofix, a map from network states to cheapest fix-action sequences, in order to prune cases where a fix-action sequence has reached a network state in a cheaper way before, *c*) and Oatt, a map from network states to optimal attack action sequences, in order to spare the search for an attack action sequence if we have already saved one.

C^0 is always equal to the cost of the cheapest fix-action sequence σ found so far which leads to a state with zero attacker success probability, i.e. $C^0 = \min\limits_{\sigma} c^\mathsf{M}(\sigma)$ such that $p^*(\mathsf{I}^\mathsf{N}[\![\sigma]\!]) = 0$. Any fix-action sequence with higher cost is dominated by definition and can thus be safely pruned.

Ofix maps each already considered network state to the cheapest fix-action sequence σ reaching this state found so far. If $\sigma' = \mathit{Ofix}(s^\mathsf{N})$ is defined in the current network state s^N and $c^\mathsf{M}(\sigma) > c^\mathsf{M}(\sigma')$, we can stop right away and prune σ as well as all successors, as σ is more expensive than the already known sequence σ' leading to the same network state. Even if this is not the case, but $\mathit{Ofix}(s^\mathsf{N})$ is defined, we can save an additional search in the attacker spate space.

Oatt maps each already considered network state to the computed optimal attack action sequence, i.e., if σ leading to $s^\mathsf{N} = \mathsf{I}^\mathsf{N}[\![\sigma]\!]$ was considered before, we store the corresponding optimal attack plan $\vec{a}$, $\mathit{Oatt}(s^\mathsf{N}) := \vec{a}$. We can similarly also make use of the optimal attack plan for the parent state of s^N. This can be done by letting σ be $f_1, \ldots, f_n$, then computing the parent state $s^\mathsf{N}_{\text{parent}} = \mathsf{I}^\mathsf{N}[\![f_1, \ldots, f_{n-1}]\!]$ and afterwards once again using the map: $\vec{a}_{\text{parent}} = \mathit{Oatt}(s^\mathsf{N}_{\text{parent}})$. Having the parent attack plan $\vec{a}_{\text{parent}}$ is useful, because we can also spare an additional search in the attacker state space if $\vec{a}_{\text{parent}}$ is still applicable to the state space induced by the current network state s^N.

The mitigation analysis algorithm PARETOFRONTIER (Figure 3) expects as arguments a network state s^N, the corresponding fix-action sequence σ leading to s^N, the sleep set for σ and the mitigation budget b^M. PARETOFRONTIER explores the space of applicable fix sequences in a *Iterative Deepening search (IDS)* manner as described in Figure 2. This means we keep executing PARETOFRONTIER with an increasing mitigation budget b^M_{curr} until a termination criterion is satisfied. We initialize b^M_{curr} to the cost of the

```
procedure IDS-ParetoFrontier()
global: $\Sigma$, $C^0$, $\mathit{Ofix}$, $\mathit{Oatt}$, cut_off
$b^{\mathsf{M}}_{curr} \leftarrow \max_{f \in \mathsf{F}} c^{\mathsf{M}}(f)$;
loop
    cut_off ← false;
    call ParetoFrontier($\mathsf{I}^{\mathsf{N}}$, $\langle\rangle$, $\emptyset$, $b^{\mathsf{M}}_{curr}$)
    if $C^0 < \infty$ then return; endif
    if not cut_off then return; endif
    if $b^{\mathsf{M}}_{curr} = b^{\mathsf{M}}_0$ then return; endif
    $b^{\mathsf{M}}_{curr} \leftarrow min(b^{\mathsf{M}}_{curr} * \gamma_{b^{\mathsf{M}}}, b^{\mathsf{M}}_0)$;
```

Figure 2. IDS-oriented algorithm for computing Pareto optimal frontier.

most expensive fix-action. We maintain the global boolean flag cut_off to indicate in ParetoFrontier that the search was cut off because of low budget. The Pareto frontier under construction Σ is initially empty and C^0 is initially equal to ∞. In each iteration, b^{M}_{curr} is increased by multiplying it with a factor $\gamma_{b^{\mathsf{M}}}$. The IDS terminates if one of the following conditions holds: 1) we have already found a state s^{N} with $p^*(s^{\mathsf{N}}) = 0$, 2) during the last call to ParetoFrontier, the search was not cut off because of low budget, or 3) we already tried the maximal budget b^{M}_0.

Theorem 2 IDS-ParetoFrontier *always terminates and computes* Σ *such that it is equal to the pareto frontier* $\mathcal{F}$ *modulo permutations on commutative fix-actions for a given mitigation-analysis task* M.

Proof: We will first argue about the correctness of ParetoFrontier and lastly about IDS-ParetoFrontier itself.

The plain ParetoFrontier algorithm without any of the optimizations would do the following: enumerate all fix action sequences σ within the budget, compute for every sequence the corresponding network state s^{N}, check whether s^{N} was already seen, compute $p^*(s^{\mathsf{N}})$ if this is not the case and finally store it in Σ unless it is dominated by another sequence σ' in Σ. The plain algorithm indeed terminates because of the duplicate check on s^{N} and the finiteness of the network state space, which we already argued in the proof to Theorem 1. Finally, it computes Σ such that is equal to $\mathcal{F}$, because all fix-action sequences in the budget (modulo duplicate states) are checked. It remains to explain why the optimizations conserve this fact. We will argue for each optimization step by step.

```
procedure ParetoFrontier($s^{\mathsf{N}}$, $\sigma$, sleep, $b^{\mathsf{M}}$)
1:  let $s^{\mathsf{N}}_{\text{parent}}$ be parent of $s^{\mathsf{N}}$ w.r.t. $\sigma$
2:  $\vec{a}_{\text{parent}} \leftarrow \mathit{Oatt}(s^{\mathsf{N}}_{\text{parent}})$
3:  if $\vec{a}_{\text{parent}}$ applicable to $\mathsf{I}^{\mathsf{A}}$ and $\mathsf{I}^{\mathsf{A}}[\![\vec{a}_{\text{parent}}]\!] \models \mathsf{G}$ then
4:      $p^*(s^{\mathsf{N}}) \leftarrow p^*(s^{\mathsf{N}}_{\text{parent}})$;
5:      $\vec{a} \leftarrow \vec{a}_{\text{parent}}$;
6:  else if $\mathit{Ofix}(s^{\mathsf{N}})$ and $\mathit{Oatt}(s^{\mathsf{N}})$ are defined then
7:      $p^*(s^{\mathsf{N}}) \leftarrow c^{\mathsf{M}}(\mathit{Ofix}(s^{\mathsf{N}}))$;
8:      $\vec{a} \leftarrow \mathit{Oatt}(s^{\mathsf{N}})$;
9:  else
10:     compute $p^*(s^{\mathsf{N}})$ and corresponding $\vec{a}$;
11: endif
12: if $p^*(s^{\mathsf{N}}) \leq \min_{\sigma' \in \Sigma}\{p^*(\mathsf{I}^{\mathsf{N}}[\![\sigma']\!]) \mid c^{\mathsf{M}}(\sigma') \leq c^{\mathsf{M}}(\sigma)\}$ then
13:     remove all $\sigma' \in \Sigma$ dominated by $\sigma$;
14:     add $\sigma$ to $\Sigma$;
15: endif
16: $\mathit{Ofix}(s^{\mathsf{N}}) \leftarrow \sigma$;
17: $\mathit{Oatt}(s^{\mathsf{N}}) \leftarrow \vec{a}$;
18: if $p^*(s^{\mathsf{N}}) = 0$ then
19:     $C^0 \leftarrow c^{\mathsf{M}}(\sigma)$;
20:     return;
21: endif
22: $\{f_1, \ldots, f_n\} \leftarrow app(s^{\mathsf{N}})$;
23: for $i = 1, \ldots, n$ do
24:     $\sigma_{\text{succ}} \leftarrow \sigma \circ \langle f_i \rangle$;
25:     if $c^{\mathsf{M}}(\sigma_{\text{succ}}) > C^0$ then continue; endif
26:     if $f_i \in sleep$ then continue; endif
27:     $s^{\mathsf{N}}_{\text{succ}} \leftarrow \mathsf{I}^{\mathsf{N}}[\![\sigma_{\text{succ}}]\!]$;
28:     if $\sigma' \leftarrow \mathit{Ofix}(s^{\mathsf{N}}_{\text{succ}})$ is defined then
29:         if $c^{\mathsf{M}}(\sigma_{\text{succ}}) > c^{\mathsf{M}}(\sigma')$ then continue; endif
30:     if $c^{\mathsf{M}}(\sigma_{\text{succ}}) > b^{\mathsf{M}}$ then
31:         cut_off ← true;
32:         continue;
33:     endif
34:     $sleep(\sigma_{\text{succ}}) \leftarrow sleep \cup$
35:                 $\{f_j \mid f_i, f_j$ are commutative$, 1 \leq j < i\}$;
36:     ParetoFrontier($s^{\mathsf{N}}_{\text{succ}}, \sigma_{\text{succ}}, sleep(\sigma_{\text{succ}}), b^{\mathsf{M}}$);
37: endfor
```

Figure 3. Depth-oriented tree search for computing Pareto optimal frontier.

1) Checking the applicability of $\vec{a}_{\text{parent}}$ in lines 2-5: Consider fix-action sequence σ leading to network state s^{N} with parent network state $s^{\mathsf{N}}_{\text{parent}}$ and plan $\vec{a}_{\text{parent}}$ for state space of $s^{\mathsf{N}}_{\text{parent}}$. We can get $\vec{a}_{\text{parent}}$ from the map Oatt which is always correctly set in line 17. Consider $\vec{a}_{\text{parent}}$ as still being applicable and leading to a goal state for s^{N} which is checked in line 3, $\vec{a}_{\text{parent}}$ is *a)* either an optimal plan for s^{N} and $p^*(s^{\mathsf{N}}) = p^*(s^{\mathsf{N}}_{\text{parent}})$ or *b)* there is another plan $\vec{a}$ optimal for s^{N} with higher success probability than $\vec{a}_{\text{parent}}$. Everything is fine in case 1a). In case 1b), $p^*(s^{\mathsf{N}}_{\text{parent}})$ rather gives us a lower bound for $p^*(s^{\mathsf{N}})$, i.e. $p^*(s^{\mathsf{N}}) > p^*(s^{\mathsf{N}}_{\text{parent}})$, which is also fine. The reason is that all we want to know is whether we can add σ to Σ which we only could if $p^*(s^{\mathsf{N}}) < p^*(s^{\mathsf{N}}_{\text{parent}})$.
2) Taking $(\sigma', \vec{a}')$ from $\mathit{Ofix}(s^{\mathsf{N}})$ and $\mathit{Oatt}(s^{\mathsf{N}})$ and not further considering σ if $c^{\mathsf{M}}(\sigma) > c^{\mathsf{M}}(\sigma')$ in lines 6-8 and 28-29 and: it is clearly not necessary to compute $\vec{a}$ again, if we have done it already for exactly the same s^{N}. It is further safe to prune σ if $c^{\mathsf{M}}(\sigma) > c^{\mathsf{M}}(\sigma')$, because for all fix-action suffixes σ_{suff}, $\sigma \circ \sigma_{\mathit{suff}}$ will always be dominated by $\sigma' \circ \sigma_{\mathit{suff}}$ if $c^{\mathsf{M}}(\sigma) > c^{\mathsf{M}}(\sigma')$.
3) Pruning fix-action sequence σ such that $c^{\mathsf{M}}(\sigma) > C^0$ in line 25: this can be done because σ is clearly dominated

by another sequence σ' already in Σ with $p^*(\mathsf{I}^\mathsf{N}[\![\sigma']\!]) = 0$ by positivity of $p^*(\mathsf{I}^\mathsf{N}[\![\sigma]\!])$. C^0 is only assigned in line 19 and only if $p^*(s^\mathsf{N}) = 0$ and $c^\mathsf{M}(\sigma) \leq C^0$. The latter is in turn enforced by the check in line 25 itself.

4) Not considering permutations of commutative fix-actions f_i and f_j by applying the sleep set method in lines 26 and 34: we can easily derive from Definition 9 that commutativity implies for all network states s^N, $s^\mathsf{N}[\![\langle f_i, f_j\rangle]\!] = s^\mathsf{N}[\![\langle f_j, f_i\rangle]\!]$. With the sleep set method, we enforce that only one ordering of f_i and f_j is considered if they are both applicable in a state s^N. Let $app(s^\mathsf{N})$ be $f_1, \dots, f_n$ and $i < j$, then we first call ParetoFrontier$(\sigma \circ \langle f_i\rangle, sleep(\sigma \circ \langle f_i\rangle))$ and f_j is not in $sleep(\sigma \circ \langle f_i\rangle))$ and can thus still be considered in the recursion. Later, we call ParetoFrontier$(\sigma \circ \langle f_j\rangle, sleep(\sigma \circ \langle f_j\rangle))$ and f_i is in $sleep(\sigma \circ \langle f_j\rangle))$ and not considered in the recursion such that we effectively only consider action sequences σ in which f_i is ordered before f_j. This preserves Pareto optimality because f_i and f_j are commutative. As a side remark: because we prune permutations of commutative fix actions, the resulting Pareto frontier Σ can only contain at most one permutation of every subset of F, even tough the permutations do not dominate each other. That is why have stated "modulo permutations on commutative fix-actions" in Theorem 2.

5) Calling ParetoFrontier in an IDS manner in Figure 2: we observe that ParetoFrontier is always called with I^N and $\langle\rangle$ in IDS-ParetoFrontier and recursive calls are constructed in lines 24 and 27 such that effectively for all calls to ParetoFrontier, s^N and σ have the relation: $s^\mathsf{N} = \mathsf{I}^\mathsf{N}[\![\sigma]\!]$. The correctness of the sleep sets is established by calling ParetoFrontier with the empty set in IDS-ParetoFrontier and the correctness of the construction in line 34. The only problem remaining with the IDS approach could be that we terminate the loop in IDS-ParetoFrontier even though Σ is not yet complete or we do not terminate at all. In fact it does not matter with which budgets b^M_{curr}, ParetoFrontier is exactly called as long as it is called with a budget large enough such that Σ is complete. We will argue why we only terminate if this is the case.
It is safe to terminate as soon as $C^0 < \infty$ since increasing b^M_{curr} can never result in finding another sequence σ with $p^*(\mathsf{I}^\mathsf{N}[\![\sigma]\!]) = 0$ and $c^\mathsf{M}(\sigma) < C^0$. Further, increasing b^M_{curr} does not change anything if there was not a single fix-action f not considered because of low budget, i.e. if cut_off = **false**. Lastly, b^M_{curr} cannot be increased if it is already equal to b^M_0.
IDS-ParetoFrontier guarantees termination because b^M_{curr} is increased in every iteration and will thus eventually be equal to b^M_0. In case, $b^\mathsf{M}_0 = \infty$, the algorithm will eventually come to a point where all reachable network states were expanded, line 29 in Figure 3 fires for all applicable f_i, cut_off will remain **false** and finally the loop in IDS-ParetoFrontier terminates.

■

6.3. Strong Stubborn Sets for Mitigation Analysis

The number n of applicable fix-actions $f_1, \dots, f_n$ branched over in a given network state s^N, cf. line 22 of Figure 3, is a critical scaling parameter as it is the branching factor in a tree search (over fix-action sequences) where each node in the search tree contains another worst-case exponential tree search for an optimal attack plan. It is therefore highly relevant to reduce n as much as possible. We next observe that, to this end, one can adapt a prominent pruning technique called *strong stubborn sets* (SSS), which allows the search to consider only a subset T_s of applicable actions in each state. SSS were invented in verification and later adapted to AI planning [51], [52]; their known variants are limited to single-agent search, like the attack planning in our setting, i. e., move-countermove setups were not considered. We provide an extension to such a setup – our setup of fix-action planning vs. attack-action planning – here. Our key observation is that, where standard SSS notions identify T_s through a subset of actions contributing to achieving the goal, we can here identify T_s through a subset of actions contributing to disvalidating the current critical attack path.

To lower the probability of a critical attack path, it is necessary to remove at least some precondition of any of its actions. In each execution of ParetoFrontier for network state s^N, we have a critical path $\vec{a}$. Based on this, we can define a 'relevant' set of propositions P which is the set of negated propositions preconditioned in $\vec{a}$, i.e. $\mathsf{P} = \{\neg p \mid p \in pre(a_i), a_i \in \vec{a}\}$. Relevant fix-actions then are ones helping to render $\vec{a}$ non-applicable; specifically, we define the set $L^\mathsf{P}_s := \{f \mid \exists\ p \in post(f) : p \in \mathsf{P}\}$ of those fix-actions that have an element from P in the postcondition. In line with previous AI planning terminology [52], we call L^P_s a *disjunctive action landmark*: a set of fix-actions L so that every applicable fix-action sequence that starts in s^N and ends in t^N where $t^\mathsf{N} \cap \mathsf{P} \neq \emptyset$ contains at least one action $f \in L$. Intuitively, a disjunctive action landmark is a set of actions at least one of which must be used to invalidate $\vec{a}$.

Now, towards identifying a subset of applicable fix-actions branching over which in a network state s^N suffices for Pareto optimality, using only L^P_s in T_s would be insufficient. This is because it is possible that no action from L^P_s is actually applicable in s^N, so we must first enable such an action. For this purpose, we define the notion of *necessary enabling set* N^f_s, as the set of fix-actions achieving a fix-action f precondition not true in s^N, i.e. $N^f_s = \{f \mid \exists\ p \in pre(f).\ p \notin s \wedge p \in post(f)\}$.

Finally, for the definition of SSS, remember the notion of interference from Definition 9 and that $\inf(f)$ is the set of fix-actions with which f interferes. We must also include interfering fix-actions into the set of fix-actions considered, because interfering actions represent alternate exclusive choices that the search needs to branch over.

Definition 10 (strong stubborn set [52]) *Let* Π *be a mitigation-analysis task with network propositions* P^N *and fix-actions* F*, let* s^N *be a network state of* Π*, let* $\vec{a}$ *be a*

```
procedure ComputeStubbornSet($s^{\mathsf{N}}$, $\mathsf{P}$)
  $T_s \leftarrow L_s^{\mathsf{P}}$; /* for some disj. action landmark $L_s^{\mathsf{P}}$ */
  repeat
    for all $f \in T_s$ do
      if $f \in app(s^{\mathsf{N}})$ then
        $T_s \leftarrow T_s \cup \inf(f)$;
      else /* for some nec. enabling set $N_s^f$ */
        $T_s \leftarrow T_s \cup N_s^f$;
  until $T_s$ reaches a fix point
  return $T_s$
```

Figure 4. Strong stubborn set comp. for state s and proposition set P.

critical attack path, and let $\mathsf{P} = \{\neg p \mid p \in pre(a_i), a_i \in \vec{a}\}$. *A* strong stubborn set *(SSS) in* s^{N} *is an action set* $T_s \subseteq \mathsf{F}$ *such that:*

1. *T_s contains a disjunctive action landmark L_s^{P} for P in s^{N}.*
2. *For each $f \in T_s \cap app(s^{\mathsf{N}})$, we have* $\inf(f) \subseteq T_s$*.*
3. *For each $f \in T_s \setminus app(s^{\mathsf{N}})$, we have $N_s^f \subseteq T_s$ for the necessary enabling set N_s^f of f in s^{N}.*

The SSS computation algorithm in Figure 4 starts with the disjunctive action landmark L_s^{P} and adds actions to the candidate set until conditions 2 and 3 are satisfied. Hence, the algorithm indeed computes a SSS. It is called in PARETOFRONTIER in line 22 before iterating over the applicable operators $app(s^{\mathsf{N}}) \subseteq \mathsf{F}$. Given the SSS T_s, it is sufficient for the algorithm to iterate solely over the operators in $T_{app(s^{\mathsf{N}})} := T_s \cap app(s^{\mathsf{N}})$ instead of the complete set $app(s^{\mathsf{N}})$, while preserving the Pareto optimality of the algorithm. This statement is formally proven in Theorem 3.

Theorem 3 *Using only $T_{app(s^{\mathsf{N}})}$ instead of app(s^{N}) in line 22 of* PARETOFRONTIER *preserves Theorem 2.*

Proof: For any state fix-action sequence σ and $s^{\mathsf{N}} = \mathsf{I}^{\mathsf{N}}[\![\sigma]\!]$, non-empty fix-action suffixes σ_{suff} which do not invalidate the attacker plan $\vec{a}$ need not be considered, as $\vec{a}$ would still be a plan for $s^{\mathsf{N}}[\![\sigma_{\text{suff}}]\!]$, but (by positivity of fix-action costs) $\sigma \circ \sigma_{\text{suff}}$ would be dominated by σ. We thus show that for all states s^{N} from which a cheapest fix-action sequence leading to a state where $\vec{a}$ is not applicable anymore and consisting of $n > 0$ actions exists, $T_{app(s)}$ contains an action starting such a sequence. A simple induction then shows that PARETOFRONTIER restricted to $T_{app(s)}$ is Pareto optimal. The rest of the proof follows that of [1, Theorem 1].

Let T_s be a SSS computed by Alg. 4 and $\sigma = f_1, \dots, f_n$ be a cheapest sequence for s invalidating $\vec{a}$. Since T_s contains a disjunctive action landmark for the propositions preconditioned by $\vec{a}$, σ contains an action from T_s. Let f_k be the action with smallest index in σ that is also contained in T_s, i.e., $f_k \in T_s$ and $\{f_1, \dots, f_{k-1}\} \cap T_s = \emptyset$. Then:

1) $f_k \in app(s)$: otherwise by definition of SSS, a necessary enabling set $N_s^{f_k}$ for f_k would have to be contained in T_s, and some action from $N_s^{f_k}$ would have to occur before f_k in σ to enable f_k, contradicting that f_k was chosen with the smallest index.
2) f_k is independent of $f_1, \dots, f_{k-1}$: otherwise, using $f_k \in app(s)$ and the definition of SSS, at least one of $f_1, \dots, f_{k-1}$ would have to be contained in T_s, again contradicting the assumption.

Hence, we can move f_k to the front: $\sigma' = f_k, f_1, \dots, f_{k-1}, f_{k+1}, \dots, f_n$ is also a sequence for s invalidating $\vec{a}$. It has the same cost as σ and is hence a cheapest such sequence. Thus, we have found a cheapest fix-plan of length n started by an action $f_k \in T_{app(s)}$, completing the proof. ■

7. Practical Model Acquisition

The formalism and algorithm introduced in the previous sections encompass a broad range of network models. In this section, we describe a highly automated approach to acquire a particular form of such network models in practice, demonstrating our method to be readily applicable. The general workflow is similar to MulVAL [38], which integrates machine-readable vulnerability descriptions and reports from network vulnerability scanners such as Nessus to derive a simple logical model specified in Datalog. 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.[4]

7.1. Workflow

From discussion with security analysts, we learned that the amount of information available for analysis is often limited initially, sometimes by nature of the assignment (white-box versus black-box tests), but mostly from the fact that evaluation of in-house software and configurations is a large part of the analysis. Choosing the hosts to focus on, however, requires the identification of critical entry points, in later stages w.r.t. mitigations already considered. We will thus describe the following work-flow, which we plan to evaluate in actual penetration testing scenarios as part of future work (depicted in Figure 5).

1) In the first step, the user scans a network using the Nessus tool, resulting in an XML report. Our current toolchain supports only network-wide scans. Nessus, as well as several OVAL interpreters [3] supports host-wise scans, which can be gathered centrally. This would give much more precise results, which can be translated in a very similar way.
2) The user describes the network topology in a JSON formatted file, including the set of hosts initially under adversarial control. If this file is not given, we assume all hosts are interconnected w.r.t. every port that appears in the Nessus report. Currently, this file has to be created by hand. Penetration testers often have

[4] Code is available at http://upload.soundadl.bplaced.net/whatif.zip

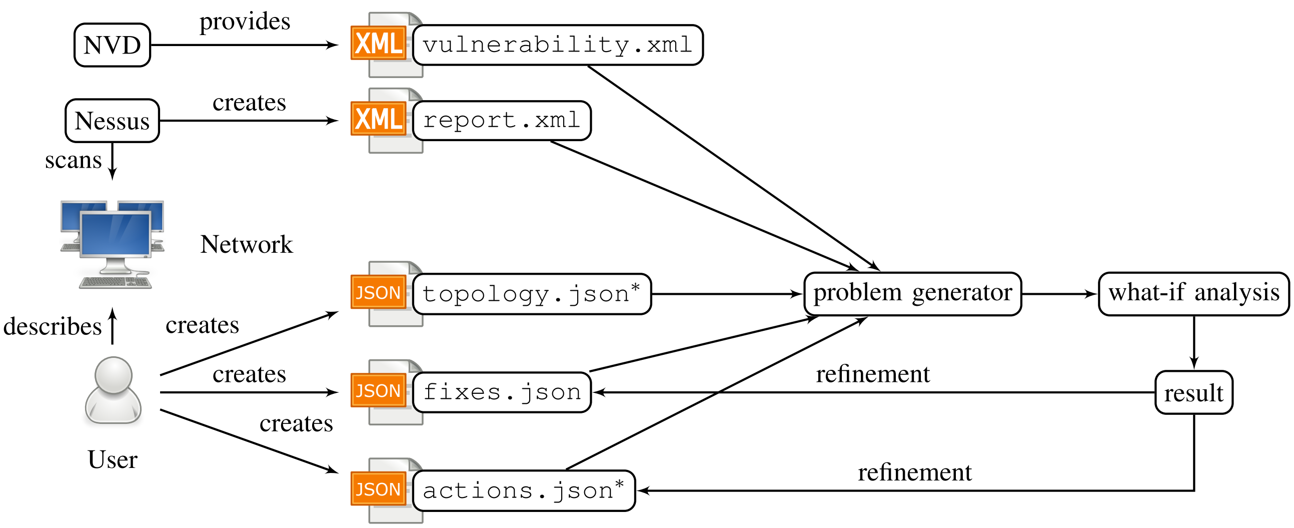


Figure 5. Workflow for model acquisition via network scanning, assuming a fixed attacker and mitigation budget. User input marked with * can be empty. The file `topology.json` can be left empty, in which case an open network is assumed.

access to firewall rules in machine-readable formats (e.g., Cisco, juniper), which could be used to derive information about connectivity between subnets. The relation between hosts and subnets may be deduced based on IP-Prefixes, or entered by hand. Hence, this part of the workflow can be mechanised easily.

3) The user specifies the fixes the analysis should consider. Initially, this list can be 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.
4) The user specifies 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. Both can be refined in subsequent steps, however, as attacker budget and action cost, as well as mitigation budget and fix-action costs are relative to one another, typically only one is modified.
5) Finally, 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 some patches might be more expensive than others (which can be reflected in the associated mitigation costs), or some firewalls proposed might be to restrictive (which can be reflected by instantiating the firewall rule). Similarly, if vulnerabilities reported by Nessus are not exploitable in this network, or priori knowledge gives a more accurate estimate about the probability of an exploitable configuration, the action model can be adjusted accordingly.
6) The refinement is repeated, until a feasible mitigation strategy is reached.

7.2. Network Topology and Vulnerabilities

Like in Example 1, the network topology is defined via network predicates $\mathsf{subnet}(z,h) \in \mathsf{I}^{\mathsf{N}}$ for every host h in subnet z, $\mathsf{haclz}(z_1, z_2, port, proto) \in \mathsf{I}^{\mathsf{N}}$ for every z_1, from where all hosts in z_2 are reachable via $(port, proto)$, which are derived from a JSON file, to allow for easy manual adjustment.

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

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

and $O(a) = \{o_{success}, o_{fail}\}$. 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, 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.

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 $type = \mathsf{integrity}$. Hence, similar to MulVal, we identify this vulnerability with a privilege escalation. The latest version 3 of the CVSS standard, released in June 2015, provides a new metric `in_scope` to specifically designate such vulnerabilities. While this metric is still not specific enough to accurately describe propagation, it at least avoids this drastic over-approximation. As of now, all vulnerability feeds provided by the NVD are classified using CVSSv2, hence we hope for quick adoption of the new standard. Consequently, and as opposed to Example 3, $pre^{\mathsf{A}}(a) = \mathsf{compromised}(h, \mathsf{integrity})$, and $post(o_{success}) = \mathsf{compromised}(h, type)$. 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.

As a first approximation, 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 $p(O_{success}) = p'$ and $p(O_{fail}) = 1 - p'$, where $post(o_{fail}) = \top$. 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.

7.3. Threat Model

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

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

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

7.4. 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 5 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 countermeasures 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.

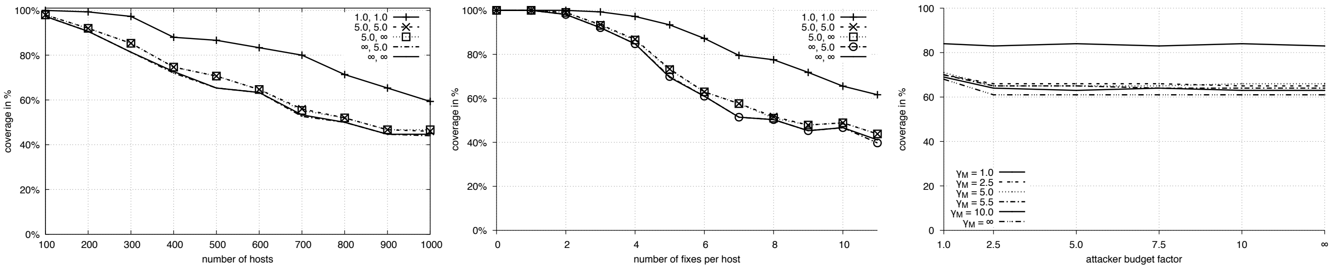
Firewall schema. There are two firewall schemas, one for firewalls between subnets, one for host-wise packet filtering. The former is defined by source and destination subnet along with port and protocol. Similar to the fix schema, any of the value may be specified, or left open as a wild card *, in which case a fix-action similar to the firewall fix in Example 5 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.

8. Experiments

To evaluate our mitigation analysis algorithm, we use a problem generator that produces network topologies and host configurations based on known vulnerabilities. For details to the generator, we refer the reader to the Appendix A. This facilitates the performance evaluation of our mitigation analysis algorithm w.r.t. several dimensions, as we can sample an arbitrary number of problems for any number of hosts, fix actions and any combination of attacker and mitigation budget. We stress, however, that this provides only insight into the applicability of the algorithm itself, not our approach as a whole. Provided the problems we generate are realistic, our results give an indication of the space of problems we can treat efficiently.

We have implemented the mitigation analysis algorithm on top of the FD planning tool [16]. 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.

In our evaluation we focus on coverage values, so 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.



(a) (b) (c)

Figure 6. % 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.

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 versas γ_{A} the factor for the attacker budget.

In Figure 6(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. The fact that the coverage decreases slower than linearly in number of hosts is promising at likely a result of stubborn set technique employed, as it restricts analysis to fix action sequences eventually effecting the attack actions in the computed critical paths which are only linearly dependent in the number of hosts.

In Figure 6(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, were 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 6(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 6(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. Again, we attribute this to the problems we generate: In many cases, the mitigation strategy that results in the minimal possible attacker success probability is cheaper than the mitigation budget resulting from $\gamma_{\mathsf{M}} = 2.5$. In almost half of the instances solved for $\gamma_{\mathsf{M}} = 2.5$, this minimal attacker success probability turned out to be 0. In these cases specifically, the mitigation analysis can readily prune mitigation strategies with higher costs, even if more mitigation budget is available, as we maintain the current bound for the cost of lowering the attacker probability to zero (C^0, see Section 6.2).

9. Conclusion & Future work

The what-if analysis mechanism 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, which we plan to evaluate on real networks in the future, 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 major challenge lies in the effective computation of the Pareto frontier. As of now, this stands and falls with the speed with which a first good solution – a cheap fix-action sequence reducing attaker success probability to a small value – is found. In case that happens quickly, our pruning methods and thus the search become highly effective; in case it does not happen quickly, the search often becomes prohibitively enumerative and exhausts our 30 minute time limit. In other words, the search may, or may not, "get lucky". What is missing, then, is effective *search guidance* towards good solutions, making it more likely to "get lucky". This is exactly the mission statement of heuristic functions in AI heuristic search procedures. The key difference is that these procedures address, not a move-countermove situation as in fix-action sequence search, but single-player (just "move") situations (like at our attack-planning level, where as mentioned we are already using these procedures). This necessitates the extension of the heuristic function paradigm – solving a *relaxed* (simplified) version of the problem, delivering relaxed solution cost as a lower bound on real solution cost – to move-countermove situations. This is a far-reaching topic, relevant not only to our research here but to AI at large, that to our knowledge remains entirely unexplored.[5] Notions of move-countermove relaxation are required, presumably over-approximating the defender's side while under-approximating the attacker's side of the game, and heuristic functions need to be developed that tackle the inherent min/max nature of the combined approximations without spending too much computational effort. For our concrete scenario here, one promising initial idea is to fix the attacker's side to the current optimal critical attack path, and setting the defender's objective – inside the heuristic function over-approximation – as reducing the success probability of that critical attack path as much as possible, while minimizing the summed-up fix-action cost. This results in an estimation of fix-action quality, which should be highly effective in guiding the fix-action level of the search towards good solutions quickly.

Regarding the practical acquisition of more refined models, the quality of the results of our analyses of course hinges on the accuracy of the input model. In practice, 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. This is partially due to the fact that vulnerabilities are often discovered in the process of pentesting, which a simulation cannot reproduce. (Although potential zero-day exploits can in principle be modeled in our framework as a particular form of attack-actions, that exist only with a given probability.) But it is also due to the fact that current vulnerabilities lack necessary information to derive these models automatically. There are two factors to the latter. First, economically, a more detailed machine-readable description of vulnerabilities requires considerable effort, hence there needs to be an incentive to provide this data. The successful commercial use of simulated pentesting at Core Security is partly the result of fine-grained data available within the company, and is partly the incentive for the further refinement of that 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, capturing 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. Similar to OVAL, which describes preconditions declaratively, but defines tests that can evaluate machines, the effect of privilege escalations can be determined on a concrete system, if a vulnerability description provides sufficient information about the type of escalation that takes place. Providing a concise, abstract and versatile model for privilege escalation is thus necessary to provide the basis for automated acquisition of realistic network models.

[5] Game-state evaluation mechanisms are of course widely used in game-playing, yet based on weighing (manually or automatically derived) state features, not on a relaxation paradigm automatically derived from the state model.

References

[1] Yusra Alkhazraji, Martin Wehrle, Robert Mattmüller, and Malte Helmert. A stubborn set algorithm for optimal planning. In Luc De Raedt, editor, *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, pages 891–892, Montpellier, France, August 2012. IOS Press.

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

[3] Jonathan Baker, Matthew Hansbury, and Daniel Haynes. The oval® language specification. *MITRE, Bedford, Massachusetts*, 2011.

[4] Mark Boddy, Jonathan Gohde, Tom Haigh, and Steven Harp. Course of action generation for cyber security using classical planning. In Susanne Biundo, Karen Myers, and Kanna Rajan, editors, *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 12–21, Monterey, CA, USA, 2005. Morgan Kaufmann.

[5] Rainer Böhme and Márk Félegyházi. Optimal information security investment with penetration testing. In *Proceedings of the 1st International Conference on Decision and Game Theory for Security (GameSec'10)*, pages 21–37, 2010.

[6] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

[7] Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors. *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*. AAAI Press, 2010.

[8] Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson. Rational choice of security measures via multi-parameter attack trees. In *1st International Workshop on Critical Information Infrastructures Security (CRITIS'06)*, pages 235–248, 2006.

[9] Ahto Buldas and Roman Stepanenko. Upper bounds for adversaries' utility in attack trees. In *Proceedings of the 3rd International Conference on Decision and Game Theory for Security (GameSec'12)*, pages 98–117, 2012.

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

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

[12] Karel Durkota and Viliam Lisý. Computing optimal policies for attack graphs with action failures and costs. In *7th European Starting AI Researcher Symposium (STAIRS'14)*, 2014.

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

[14] Nirnay Ghosh and S. K. Ghosh. An intelligent technique for generating minimal attack graph. In *Proceedings of the 1st Workshop on Intelligent Security (SecArt'09)*, 2009.

[15] Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91)*, pages 332–342, 1991.

[16] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[17] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 162–169. AAAI Press, 2009.

[18] Malte Helmert and Hauke Lasinger. The Scanalyzer domain: Greenhouse logistics as a planning problem. In Brafman et al. [7], pages 234–237.

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

[20] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

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

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

[23] Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. In *Managing Cyber Threats: Issues, Approaches and Challenges*, chapter 5. 2005.

[24] Sergio Jimenez, Andrew Coles, and Amanda Smith. Planning in probabilistic domains using a deterministic numeric planner. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSig'06)*, 2006.

[25] Alexander Koller and Jörg Hoffmann. Waking up a sleeping rabbit: On natural-language sentence generation with ff. In Brafman et al. [7].

[26] Alexander Koller and Ronald Petrick. Experiences with planning for natural language generation. *Computational Intelligence*, 27(1):23–40, 2011.

[27] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. ADTool: security analysis with attack-defense trees. In *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST'13)*, pages 173–176, 2013.

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

[29] Dmytro Korzhyk, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *J. Artif. Int. Res.*, 41(2):297–327, May 2011.

[30] Viliam Lisý and Radek Píbil. Computing optimal attack strategies using unconstrained influence diagrams. In *Pacific Asia Workshop on Intelligence and Security Informatics*, pages 38–46, 2013.

[31] Ian Little and Sylvie Thiebaux. Probabilistic planning vs replanning. In *ICAPS Workshop on the International Planning Competition: Past, Present and Future*, 2007.

[32] Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

[33] Jorge Lucangeli, Carlos Sarraute, and Gerardo Richarte. Attack planning in the real world. In *Proceedings of the 2nd Workshop on Intelligent Security (SecArt'10)*, 2010.

[34] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In *Proceedings of the 8th International Conference on Information Security and Cryptology (ICISC'05)*, pages 186–198, 2005.

[35] Margus Niitsoo. 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)*, pages 354–370, 2010.

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

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

[38] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. Mulval: A logic-based network security analyzer. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM'05, pages 8–8, Berkeley, CA, USA, 2005. USENIX Association.

[39] Judea Pearl. *Heuristics*. Morgan Kaufmann, 1984.

[40] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the New Security Paradigms Workshop*, 1998.

[41] Ronald W. Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *IEEE Symposium on Security and Privacy*, pages 156–165, 2000.

[42] Wheeler Ruml, Minh Binh Do, Rong Zhou, and Markus P. J. Fromherz. On-line planning and scheduling: An application to controlling modular printers. *Journal of Artificial Intelligence Research*, 40:415–468, 2011.

[43] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. POMDPs make better hackers: Accounting for uncertainty in penetration testing. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, pages 1816–1824, Toronto, ON, Canada, July 2012. AAAI Press.

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

[45] B. Schneier. Attack trees. *Dr. Dobbs Journal*, 1999.

[46] Oleg Sheyner, Joshua W. Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*, pages 273–284, 2002.

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

[48] Marcel Steinmetz, Jörg Hoffmann, and Olivier Buffet. Goal probability analysis in mdp probabilistic planning: Exploring and enhancing the state of the art. *Journal of Artificial Intelligence Research*, 57:229–271, 2016.

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

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

[51] Antti Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4):297–322, 1992.

[52] Martin Wehrle and Malte Helmert. About partial order reduction in planning and computer aided verification. In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12).* AAAI Press, 2012.

[53] Sung Wook Yoon, Alan Fern, and Robert Givan. FF-Replan: a baseline for probabilistic planning. In Mark Boddy, Maria Fox, and Sylvie Thiebaux, editors, *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 352–359, Providence, Rhode Island, USA, 2007. Morgan Kaufmann.

Appendix

1. Scenario Generator

The problems we generate are modelled exactly as described in Section 7. We hence describe the scenario generation in terms of the topology, i. e., subnet relations defined by the network proposition subnet and connections described by the network proposition haclz, and the assignment of configurations, i.e., the network proposition vul_exists and corresponding actions.

Topology. The network topology generation follows previous works on network penetration testing task generators [43]. Similar to the running example, we generate networks which are partitioned into four zones: users, DMZ, sensitive and internet. The internet consists of only a single host which is initially under adverserial control, and which is connected to the DMZ zone. The DMZ and the sensitive zone each constitute a subnet of hosts, both subnets being connected to each other. The user zone is an hierarchy, tree, of subnets, where every subnet is connected to its parent and the sensitive part. Additionally, the root subnet of the user zone is also connected to the DMZ zone. A firewall is placed on every connection between subnets. While the firewalls inside the user zone are initially empty, i. e., they do not block anything, the firewalls located on connections between two different zones only allow traffic over a fraction of ports. The ports blocked initially are selected randomly. The size of the generated networks is scaled through parameter H determining the overall number of hosts in the network. To distribute H hosts to the different zones, we add for each 40 hosts one to DMZ, one to the sensitive zone, and the remaining to the user zone (cf. [43]).

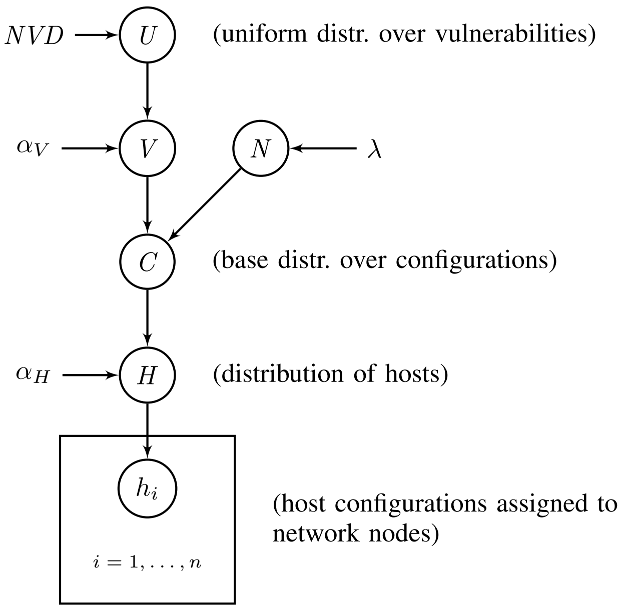

NVD
U
(uniform distr. over vulnerabilities)
α_V
V
N
λ
C
(base distr. over configurations)
α_H
H
(distribution of hosts)
h_i
$i = 1, \ldots, n$
(host configurations assigned to network nodes)

Figure 7. Probabilistic graphical model for the distribution of configurations within the network.

Configurations. Now we come to the assignment of configurations, i.e., set of vulnerabilities to hosts. In many corporate networks, host configurations are standardized, e.g., workstations have equivalent configurations or each machine within a cluster is alike. To this end, we model the distribution of the totality of hosts used in the network by means of a (nested) *Dirichlet process*.

Depending on the concentration parameter α_H, the ith host H_i is, with probability $\alpha_H/(\alpha_H + n - 1)$, drawn freshly from the distribution of configurations C, which we will explain in the followup, or otherwise uniformly chosen among all previous H_j, $j = 1, \ldots, i - 1$. Formally, $H \mid C \sim \mathrm{DP}(\alpha_H, C)$. Configurations are drawn using the following process. First, the number of vulnerabilities a configuration has in total is drawn according to a Poisson distribution, $N \sim \mathrm{Pois}(\lambda_V)$. This number determines how many vulnerabilities are drawn in the next step by means of a second Dirichlet process. This models the fact that the software which is used in the same company tends to repeat across configurations. The base distribution over which the Dirichlet process chooses vulnerabilities is the uniform distribution over the set of vulnerabilities in our database NVD, i.e., $V \sim \mathrm{DP}(\alpha_V, U_{NVD})$.

A configuration C is now chosen by drawing n from N

and then drawing n samples from V, i.e.,

$$\Pr[C = (c_1, \ldots, c_n)] = \Pr[\, V = n, c_1 = V_1, \ldots, c_n = V_n \,].$$

where $V_1, \ldots, V_n \sim V$. Observe that C are conditionally independent given $\mathrm{DP}(\alpha_V, U_{NVD})$, hence C may define the base distribution for the above mentioned Dirichlet process. Now the configurations are drawn from the distribution we just described, i.e., $h_1, \ldots, h_n \sim C$.

We are thus able to control the homogeneity of the network, as well as, indirectly, via α_V, the homogeneity of the software configurations used overall. For example, if α_H is low but α_V is high, many configurations are equal, but if they are not, they are likely to not have much intersection (provided NVD is large enough). If α_V is low but α_H is high, many vulnerabilities reappear in different host configurations.

Mitigation model. Akin to Section 7, we consider two different types of fix-actions: closing open ports by adding rules to firewalls, and closing vulnerabilities through applying known patches. For the former, we generate for each subnet and each port available in this subnet a fix-action f that in effect blocks all connections to this subnet over this port, by negating the corresponding haclz propositions. Such fix-actions are always generated for all user subnets. DMZ and sensitive conceptionally do not allow closing all ports, as some ports must remain opened for services running in those subnets. Hence, for DMZ and the sensitive zone, we randomly select a subset of open ports which must not be locked out through firewall rules, and firewall fix-actions are then only generated for the remaining ports. Patch fix-actions are drawn from the set of possible patches described inside the OVAL database that is provided from Center for Internet Security.[6] In OVAL, each patch is described in terms of an unique identifier, human readable metadata, and a list of vulnerabilties that are closed through the application of this patch. We assign patch actions to each generated configuration. Similar to before, first, the number of patches a configuration has in total is drawn according to a Poisson distribution, $N_F \sim \mathrm{Pois}(\lambda_F)$. For each configuration, first the actual number of patches n_F is sampled from N_F, and then n_F patches are drawn uniformly from the set of patches given by OVAL which affect at least one vulnerability for this configuration.

[6] https://oval.cisecurity.org/repository