Simulated Penetration Testing: From “Dijkstra” to “Turing Test++”

Jörg Hoffmann
Saarland University
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de

Abstract

Penetration testing (pentesting) is a well established method for identifying security weaknesses, by conducting friendly attacks. Simulated pentesting automates this process, through designing a *model* of the system at hand, and using model-based *attack planning* to generate the attacks. Classical planning variants of this idea are being used commercially by the pentesting industry since 2010. Such models can pinpoint potentially dangerous combinations of known vulnerabilities, but ignore the incomplete knowledge characteristic of hacking from the attacker’s point of view. Yet, ideally, the simulation should conduct its attacks the same way a real attacker would. Hence the ultimate goal is much more ambitious: *to realistically simulate a human hacker*. This is a grand vision indeed; e. g., the classical Turing Test can be viewed as a sub-problem. Taking a more practical perspective, the simulated pentesting model space spans a broad range of sequential decision making problems. Analyzing prior work in AI and other relevant areas, we derive a systematization of this model space, highlighting a multitude of interesting challenges to AI sequential decision making research.

Introduction

Penetration testing, short *pentesting*, identifies IT system security weaknesses by conducting friendly attacks. The method is well established, and several commercial tools are available (see e. g. (Burns et al. 2007)). Given the size of modern systems (like the computer networks of large internet companies), the dynamics of the security domain, as well as the costs associated with human pentesting, computer support has shifted into focus since the late 90s. The most ambitious idea, which we baptize *simulated pentesting* here, is to completely automate the pentest: Design a *model* of the system at hand, and use model-based *attack planning* to generate the attacks. The *Core Security* company (http://www.coresecurity.com/) employs this idea commercially since 2010, in their *Core IMPACT* tool, consulted by the author and using a variant of the author’s Metric-FF system (Hoffmann 2003) as the base planner.

Historically, the idea of pentesting automation originates in the consideration of so-called *attack graphs* (e. g. (Phillips and Swiler 1998; Lippmann and Ingols 2005)). The different approaches in that area differ considerably in scope and purpose, but they all model individual attack actions in terms of preconditions and postconditions. The application was first introduced to AI planning by Boddy et al. (2005), resulting in the inclusion of the *CyberSecurity* domain in the 2008 International Planning Competition. That domain considers insider attacks, and incorporates aspects of the physical world such as looking over someone’s shoulder. The Core Security model, introduced several years later (Lucangeli, Sarraute, and Richarte 2010), considers outside attackers – *hackers* – instead, and encodes network security at a technical IT-level where individual “attack actions” correspond to known exploits of software vulnerabilities.

Our approach in this paper is to start from the Core Security model and application, motivated by its commercial success, and primarily taking a technical network security perspective. We examine where this should be taken in the future to more comprehensively achieve the goals of simulated pentesting. The author believes that this endeavor has the potential to become a major driving force and application for AI sequential decision making research.

In practice, the main added value of Core IMPACT’s attack planning, and arguably of simulated pentesting in general, is *finding* relevant attacks (as opposed to actually *executing* them). The tool guides the attention of human security officers, pointing out where, in an overwhelmingly large system, the most critical security issues may lie. Hence *accuracy* is a critical asset: Accurate hints are useful, inaccurate hints are a waste of security officers’ time. According to Core Security practitioners, the ideal tool would point out the same attacks a real attacker would attempt. In other words, our vision is *to realistically simulate a human hacker*.

In its full scope, this vision is daunting. In “socio-technical attacks” (like “spear-phishing” emails, e. g. (Jagatic et al. 2007)), the classical Turing Test may appear as a sub-problem, when communicating in a social network during a reconnaissance phase.[1] Even for purely technical attacks, realistically simulating a hacker arguably is “AI-complete”, requiring among many other things to model commonsense knowledge (e. g., for password guessing).

[1] In fact, Huber et al. (2009) use the ALICE chatbot (http://alice.pandorabots.com/) for communicating with human users in a simple Facebook social engineering bot.

Approaching the problem from a practical point of view, we herein concentrate on fragments of simulated pentesting supported by existing sequential decision making formalisms. We asume that our tool will be used at model-level only, i. e. the attack plans will not actually be executed so we are facing an offline problem. Our aim is to systematize the model space. We identify two major dimensions characterizing that space: **(A) how to handle the uncertainty from the point of view of the attacker**, and **(B) the degree of interaction between individual attack components**.

Regarding (A), deterministic models like Boddy et al.'s, the Core Security model, and attack graphs, abstract from that uncertainty completely (some give actions with "smaller success probability" a higher cost). An opposite extreme, suggested by the author and co-workers (Sarraute, Buffet, and Hoffmann 2012), is a POMDP model explicitly incorporating the attacker's initial knowledge, and the knowledge gained by observations during attack execution.

Dimension (B) is less immediately visible, revealing itself only upon closer inspection of the Core Security model as well as a variety of attack graph formulations. Our conclusion will be that network graph distance, as well as delete-relaxed action models, are both relevant alternatives to the full factored action models common in AI planning. Hence, in our formulation, simulated pentesting ranges all the way from Dijkstra on an explicit graph to solving a factored POMDP, with several new special cases in between.

To familiarize the reader with simulated pentesting, and to illustrate the extreme ends of the modeling space, we start by introducing the Core Security and POMDP models. We then introduce a middle ground between these two extremes, based on an MDP abstraction of the POMDP model. We discuss the attack graph literature, as well as some related models, and systematize our findings in the form of a model taxonomy. We close the paper with a discussion of research challenges arising from that systematization, and from the simulated pentesting application at large.

Classical Planning: The Core Security Model

A detailed description of Core Security's application design is out of scope (and would violate confidentiality). But the core planning model itself has been published (Lucangeli, Sarraute, and Richarte 2010) and is easily explained. We will henceforth refer to this model as *CoreSec-Classical*. Have a look at the PDDL action schema in Figure 1.

```
(:action HP_OpenView_Remote_Buffer_Overflow_Exploit
   :parameters (?s - host ?t - host)
   :precondition (and (compromised ?s)
      (connected ?s ?t)
      (has_OS ?t Windows)
      (has_OS_edition ?t Professional)
      (has_OS_servicepack ?t Sp2)
      (has_OS_version ?t WinXp)
      (has_architecture ?t I386)
      (has_service ?t ovtrcd))
   :effect (and (compromised ?t) (increase (time) 10)))
```

Figure 1: An action schema in CoreSec-Classical. (Slightly simplified for presentation.)

Each action schema corresponds to an *exploit* of a known *vulnerability*. In our example, the exploit creates a buffer overflow in an HP OpenView service in a particular configuration of Windows. The object parameters are the source and target *hosts* (machines in the network), ?s and ?t. The hacker must already have gained control over ("compromised") ?s, and, when applying the action, gains control over ?t. Other exploits, i. e. other action schemas in the model, are different only with respect to the "has_*" preconditions, and the effect on "time" (regarding which: see below). The PDDL problem files specify host connectivity, i. e. the *network graph* whose nodes are the hosts and whose edges are their connections, as well as the *host configurations* via the "has_*" predicates. One host (modeling the internet) is compromised in the initial state, and the goal is to compromise one or several other hosts. The value of "time" should be minimized.

Towards model space systematization, note the following simplifying assumptions inherent in CoreSec-Classical:

(i) **Known network graph:** No uncertainty about the network graph topology.

(ii) **Known host configurations:** No uncertainty about the host configurations.

(iii) **Static network:** Neither host connectivity nor host configurations are affected by the actions.

(iv) **Monotonic actions:** The attacker can only gain *attack assets* (here: compromised hosts; more generally: anything an attacker may rely on during an intrusion). Once obtained, an attack asset cannot be lost.

(v) **Actions = hops:** The non-static precondition and effect of each action corresponds exactly to a "hop" in the network graph, compromising a new target host starting from an already compromised source host.

Assumptions (i) and (ii) pertain to our dimension (A), uncertainty from the point of view of the attacker. Both can be relaxed in principle, in the models we propose; our main focus will be on relaxing (ii) which appears to be more feasible in practice. Assumptions (iii-v) pertain to our dimension (B), the degree of interaction between individual attack components: These properties of CoreSec-Classical, along with our observations regarding the attack graph literature below, motivate the consideration of these restrictions as relevant special cases of pentesting action models. Note that (v) implies each of (iv) and (iii); we have separated the properties here to highlight the relevant distinction lines.

Analyzing (i–v) in CoreSec-Classical and its practical use, consider first dimension (A). Clearly, assumptions (i) and (ii) are made by the model (this information is specified in the initial state), although clearly they are not realistic from the point of view of an outside attacker. Indeed, assumption (ii) is not even realistic from the point of view of Core Security's pentesting tool: While the network graph is known, it is impossible to keep track of the configuration of every host in the network, maintained by individual users. Core IMPACT employs an *information gathering* phase, running comprehensive network scans prior to starting the pentests. The most plausible configuration is then output in the above PDDL format for the planner.[2] Core

[2] The word "plausible" is chosen intentionally here: this configuration selection is not based on a formal/probabilistic model.

IMPACT also selects a *success statistic* for each exploit, i. e., a statistic on the fraction of times an exploit typically works on a host with the observed configuration parameters. But, rather than using these statistics as probabilities (an approach we will get to later on), they are used as part of the "time" effect, which is actually an action cost amalgamating execution time and success statistic into a single number.

Regarding dimension (B), observe first that the actions are *delete-free*, i. e., we have assumption (iv). While this may be surprising from an AI planning point of view, it is actually a widely employed assumption in security models, specifically attack graphs. Intuitively, monotonicity makes sense when viewing the attack as an accumulation of assets, like access rights or information (e. g. passwords) that, once gained, are not typically lost again. (Unless the attack is detected and ends altogether which is outside our models here; we say a few words on adversarial models further below.)

Host connectivity and configurations are given in the PDDL initial state, and are not affected by the actions, so we have assumption (iii). This is a limiting assumption, as an attacker may change configuration parameters on a host in order to enable another exploit and thus access rights escalation. But that is not the most frequent attack pattern, and one can capture it by modeling different host configurations and access rights as different hosts. So *assumption (iii) is benign*. In the specific setting of CoreSec-Classical, though, (iii) entails (v): Once static predicates are removed, we end up with a grounded encoding where every action has one positive ("compromised") precondition and one positive ("compromised") effect. Consider the graph G whose nodes are all facts and whose edges (p, q) correspond to actions with precondition p and effect q. Then G is like the network graph, except that the edges now correspond to hops from host to host, so the number and cost of edges between connected hosts varies depending on the number of applicable exploits. If the planning goal consists of a single fact, i. e. there is a single goal host which is a realistic pentesting scenario, then optimal planning is equivalent to finding a shortest path from the initial host to the goal host in G.[3] In that sense, CoreSec-Classical is merely a declarative way of representing the graph G. We will henceforth refer to this simple setting as an *explicit network graph* model.

Given this, it may be surprising that Core Security uses a PDDL model and solver at all. The main added values on their side (apart from being able to handle multiple goal hosts if desired) are reduced development cost and time using an off-the-shelf tool, effortless extensibility to more general action models, as well as a few extra features on top of Metric-FF to find not one solution, but several ones for the human to consider ("optimality" relative to a crude model like this is relevant, but not reliable, in practice).

Towards Accuracy: POMDP Models

Approaching simulated pentesting from an academic rather than practical perspective, the author and co-workers (Sarraute, Buffet, and Hoffmann 2011; 2012) explored the opposite end of the modeling spectrum. Our driving question was, *what problem is Core Security actually trying to solve?* CoreSec-Classical clearly is a pragmatic proxy for a more complex problem. How to formulate that problem accurately, using standard formalisms?

Our key design decision is to model the attacker's incomplete knowledge about the network as an uncertainty of *state*: A probability distribution over possible network graphs and host configurations. An initial probability distribution encodes the knowledge prior to the attack, and observations (from explicit sensing actions and observed action outcomes) during attack execution refine that knowledge. This naturally leads to a POMDP model. Different variants of POMDPs (see e. g. (Monahan 1982; Kaelbling, Littman, and Cassandra 1998)) may be suitable. In what follows, we mainly focus on the construction of states and the behavior of actions, basic aspects that would plausibly be shared by any POMDP model of simulated pentesting. We assume a *Factored POMDP* perspective (e. g. (Hansen and Feng 2000)), which is natural and ties in well with the other models discussed herein. We distinguish the most direct extension of CoreSec-Classical, which we refer to as *CoreSec-POMDP*, from more general POMDP pentesting models.[4]

CoreSec-POMDP still (i) assumes that the network graph is known, and (v) limits actions to network graph hops; hence (as (v) implies (iii) and (iv)) the only assumption *not* made anymore is (ii). It makes two new assumptions relevant only for models incorporating uncertainty:

(vi) **Succeed-or-nothing:** Each exploit has only two possible outcomes, *succeed* or *fail*. The latter outcome has an empty effect.

(vii) **Configuration-deterministic actions:** The outcome of every action (exploit as well as sensing) depends deterministically on the network configuration.

The rationale behind (vi) is that, if an exploit fails, then the attack status does not change - the attacker does not gain anything, but does not lose anything either. This is an abstraction as some exploits may have detrimental side effects, like crashing the target host. The rationale behind (vii) is that an exploit succeeds iff the target host has the required configuration. If we execute the same exploit twice in the same configuration, then the outcome will be the same. This is an abstraction as for some exploits the outcome may depend on the processes presently running on the target host, or on other details beyond any reasonable state model. Such dependencies are infrequent though, so, like assumption (iii), *assumption (vii) is benign*. For future reference, observe that assumptions (iii) and (vii) together imply that repeating an action leads to the same state so is redundant.

States s describe the network configuration, as well as the status of the attack. In CoreSec-POMDP, states are described with the same "connected", "has_*" and "compromised" predicates as before. More general formulations may

[3] For a fixed-size goal, optimal planning is still tractable (Bylander 1994). Else, optimal planning is equivalent to the Steiner Tree Problem on G (Keyder and Geffner 2009).

[4] Our current model (Sarraute, Buffet, and Hoffmann 2012) is like CoreSec-POMDP except that we allow detrimental side effects crashing the target host, and that the model is ground/enumerative based on SARSOP (Kurniawati, Hsu, and Lee 2008).

add whatever other kinds of attack assets (software installed, passwords obtained), and/or dynamic properties (crashed machines), are deemed relevant. The initial belief b_0, in general, contains any (factored description of a) probability distribution over valuations to the state predicates. In CoreSec-POMDP, we make assumption (i) so b_0 fixes the network graph and contains a probability distribution only over the host configurations, i. e., the "has_*" predicates.

Exploits and sensing actions may in general take arbitrary forms, i. e., come with arbitrary transition and observation probabilities. In models making assumptions (vi) and (vii), exploits can be described exactly as in Figure 1, i. e., *with a conjunctive precondition and effect*, interpreted as follows. Consider a transition from s via a to s', where a is an exploit action. Distinguish whether (1) a's precondition holds in s, and s' is the outcome of a's effect applied to s; or (2) a's precondition does not hold in s, and $s = s'$. Then the transition probability $P(s'|s,a)$ is 1 in cases (1) and (2), and is 0 elsewhere. The observations are "success" in case (1), "fail" in case (2), and none elsewhere.

In CoreSec-POMDP, we restrict to exactly the same exploit descriptions as in CoreSec-Classical, but interpreted in this fashion. This is still an *explicit network graph* model, in the sense that the state-changing actions have a single non-static positive "compromised" precondition, and a single positive "compromised" effect. In other words, the model is still exclusively concerned with hops from host to host in the network. In particular, the model is still *monotonic*, as is any model with more general attack assets that cannot be lost. If we incorporate detrimental side effects, like exploits that under particular conditions crash the target host, then we are in the case of more general Factored POMDPs.

As an example of a sensing (reconnaissance) action, fitting assumption (vii) and thus CoreSec-POMDP, consider the OS (operating system) detection action in Figure 2.

```
(:action OS_Detect
   :parameters (?s - host ?t - host)
   :precondition (and (compromised ?s) (connected ?s ?t))
   :observe (and
      (when (has_OS ?t Windows2000) ("win"))
      (when (has_OS ?t Windows2003) ("win"))
      (when (has_OS ?t WindowsXPsp2) ("winXP"))
      (when (has_OS ?t WindowsXPsp3) ("winXP")))
```

Figure 2: An OS detection action. (Using a hypothetical PDDL-like syntax, in line with Figure 1.)

This sensing action allows to distinguish earlier Windows versions from WindowsXP. The sensing is imprecise, as many network reconnaissance actions are, but not in terms of different possible outcomes for the same configuration (the action is state-deterministic), but rather in terms of the information returned. The typical output is a *list* of possible configuration parameters, modeled here by returning the same observation for Windows2000 and Windows2003, as well as for different service packs of WindowsXP.

An attack is now a policy for the POMDP, choosing actions based on past observations and actions. We will discuss optimization criteria in detail below, along with our model taxonomy. To say a few words right now, observe that infinite looping behaviors are not useful in pentesting. There is only a finite number of things to achieve (hosts to compromise), and it is natural to assume that every action has a strictly positive cost. A practical pentest terminates either when reaching the goal, or when the attacker decides to give up. It is, therefore, natural to assume an absorbing terminal state and frame pentesting as a *Stochastic Shortest Path* problem. In our current POMDP model, we maximize non-discounted reward, with non-negative rewards for newly compromised hosts and strictly negative rewards for actions; a "give-up" action leads to an absorbing state.

The POMDP approach is nice in that it captures many relevant properties of real-world hacking, as far as a discrete transition system formulation allows. On the other hand, the computational issues with POMDPs are widely known. In our experiments (Sarraute, Buffet, and Hoffmann 2012), models for more than one host were infeasible and we instead devised a decomposition approach using POMDPs on single-host problems only. Also, the model itself is not easy to come by. How to design the "initial belief"? Ideally, we need to capture the knowledge of (different kinds of) real-world hackers. One approach could be to use the outcome of common network reconnaissance scanning scripts, similar to Core IMPACT's information gathering phase. We believe this is promising, and will refer to it below, but overall the question of how to obtain b_0 is still wide open.

The MDP Middle Ground

MDPs are a natural candidate for a middle ground between classical planning and POMDPs. To the author's knowledge, there is no published work on the use of MDPs as simulated pentesting models in our sense. The single exception is a STAIRS starting researcher symposium paper (Durkota and Lisý 2014), which formulates a variant of attack graphs as MDPs; we get back to this further below. In what follows, we discuss MDP simulated pentesting models in the light of the distinction lines (i–vii) identified above.

The basic idea is to *formulate the attacker's uncertainty in terms of action outcomes, hiding the details – the host configurations – that actually govern these outcomes*. This can be understood as an abstraction of the POMDP model we just discussed. The actions are as before, minus the sensing. The possible outcomes of exploits are as before, except that each outcome o of action a is assigned a probability $p(a,o)$ that does not depend on host configuration predicates.

In principle, none of (i–vii) are necessary in this context. If we do start from a POMDP model with assumptions (i) known network graph, (iii) static network, (vi) succeed-or-nothing, and (vii) configuration-deterministic actions, then the action outcome probabilities $p(a,o)$ can be obtained in a very natural manner from commonly used success statistics.

Under assumption (vii), the outcome of an exploit depends only on the network configuration. So whether or not outcome o of action a occurs can be characterized in terms of a condition $\phi(a,o)$ on the configuration predicates. Under assumption (iii), that part of the POMDP state is static. Given this, it is natural to define the probability $p(a,o)$ as the probability of $\phi(a,o)$ being true in the POMDP initial belief b_0. This still is an abstraction (see our discussion be-

```
(:action HP_OpenView_Remote_Buffer_Overflow_Exploit
    :parameters (?s - host ?t - host)
    :precondition (and (compromised ?s) (connected ?s ?t))
    :effect (and (probabilistic 0.3 (compromised ?t))
        (increase (time) 10)))
```

Figure 3: The exploit model from Figure 1, abstracted into a probabilistic-outcomes model. The "has_*" configuration preconditions are replaced by the success probability 0.3, the probability that these preconditions hold on hosts with the same information-gathering profile.

low), but a more benign one than in the general case where $\phi(a, o)$ could be affected during execution.

Under assumption (i), $\phi(a, o)$ concerns only the host configuration. Under assumption (vi), as a can only succeed or fail, the only probability we need to define is the *success probability* $p(a)$ of a in b_0. Recall now the idea to define b_0 through the outcome of comprehensive network scanning as in Core IMPACT's information gathering phase. Then the success probability of a in b_0 corresponds (modulo the precise definition of b_0 in this context, which is an open question) to Core IMPACT's aforementioned *success statistic*, measuring the fraction of times an exploit typically succeeds on a target host with the observed configuration parameters. In other words, *the probabilities we need are already there!* We simply use Core Security's success statistics, a source of action cost in CoreSec-Classical, as success probabilities instead. (Such probabilities have been explored in preliminary work at Core Security already, in an "attack-tree" approach (Sarraute, Richarte, and Obes 2011), see next section.)

In the resulting MDP model, an initial state s_0 encodes the outset of the attack, and we wish to find a (closed partial) policy for s_0. The state predicates consist of the network graph and whatever dynamic properties are being modified by the exploits. Any action conditions on host configuration predicates are removed; by (iii), action effects on these are not present. For illustration, our previous exploit model from Figure 1 thus changes to the one shown in Figure 3.

If the MDP includes *only* actions like that in Figure 3, i. e., if we start from the CoreSec-POMDP model, then we obtain an MDP making assumption (v) actions = hops. As (v) implies (iv), with (i,iii,vi,vii) already made above, only (ii) remains relaxed. The state predicates are the same as before minus "has_*", i. e., we only have "connected" and "compromised" of which the former is static. We refer to this direct abstraction of CoreSec-POMDP, respectively direct extension of CoreSec-Classical, as *CoreSec-MDP*.

With respect to the POMDP model, as we have seen, the MDP approach dramatically simplifies finding the probabilities. It of course reduces computational complexity. The prize we pay is a loss of accuracy, incurred by what can be viewed as the loss of ability to gather knowledge. Say that, as described above, we set $p(a)$ to the likelihood of a's configuration preconditions holding in b_0. Then, in b_0, the success probabilities in the POMDP and MDP are the same. However, in the POMDP, as we make observations, we refine our knowledge about the network and hence the success prospects of exploits. In the MDP, $p(a)$ never changes.

One consequence of this is an independence assumption. In the POMDP model, executing exploit a yields information about other exploits a' relying on similar configuration properties. If we already know that a succeeds, chances are higher that a' succeeds as well, and vice versa. By contrast, $p(a)$ and $p(a')$ are separate in the MDP model, and handled like independent probabilities.

Another, more dire, consequence is that the MDP trivializes goal probability. Consider the exploit from Figure 3, and say that the target host ?t does not actually have the configuration required for this particular exploit to work. Nevertheless, if we apply the exploit arbitrarily often, the chance of breaking into ?t eventually is 1. This turns the exploit into a kind of dice throw, "waiting for the right outcome". Such behavior certainly feels stark and unintuitive; more precisely, it contradicts our assumptions (iii) and (vii).

Given that (vii) exploit outcomes depend only on the network configuration, and (iii) this configuration is static, as pointed out previously there is no point in trying the same exploit twice on the same target. While (iii) and (vii) are abstractions, they do capture the most typical behavior (which is why contradicting them feels stark and unintuitive). Assuming (iii) and (vii) as given, a straightforward solution to the "waiting for the right outcome" issue is the:

(viii) **Apply-once constraint:** Allow to apply each exploit, on each target host, at most once.

In principle, we can impose (viii) at PDDL level, e. g. introducing a flag "(attemptValid HP_OpenView ?t)" into the precondition and delete list in Figure 3. To distinguish models that are monotonic *except* for the apply-once constraint, we will view this constraint as part of the model semantics, not of the action descriptions themselves. (The understanding being that, to preserve the Markov property, the subset of still-available actions is part of the state.)

Attack Graphs and Other Models

With the above details in mind, let's take a step back and consider related models for pentesting computer support.

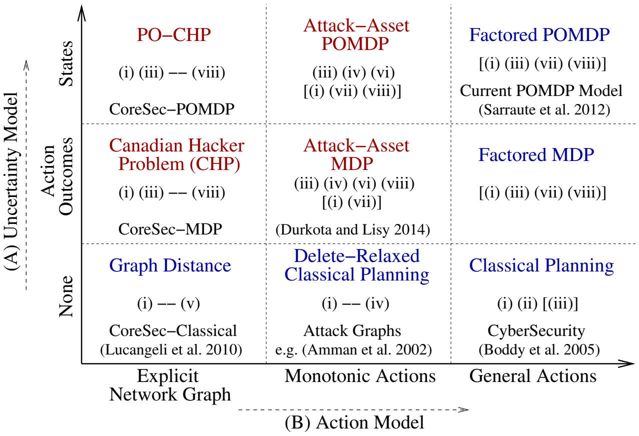
Attack graphs, first introduced by Philipps and Swiler (1998), are the model most closely related to a planning-based formulation of pentesting. Attacks are broken down into atomic components, often referred to as actions, described by conjunctive preconditions and postconditions over relevant properties of the system under attack. In other words, the action model is basically STRIPS. The attack graph represents some form of analysis of threats that arise through possible action combinations, i. e., reachability given these actions. This is, again, much as in STRIPS planning, except that many works aim to find, not just one attack plan, but an overview of all possible (or most relevant) attack plans, for the consideration of human security officers. Due in part to this attack-overview ambition, attack graphs come in many different variants, and the term "attack graph" is heavily overloaded. From our point of view here, the most relevant distinction lines are as follows.

In several early works (e. g. (Schneier 1999; Templeton and Levitt 2000)), the attack graph is the action model itself, presented to the human as an abstracted overview of (atomic) threats. With impetus from researchers with a model checking background, it was then proposed to instead reason about combinations of atomic threats, where the "attack graph" (sometimes "full" attack graph) is the state space (e. g. (Ritchey and Ammann 2000; Sheyner et al. 2002)). Then, quite early on in the history of attack graphs, it was suggested that monotonic formulations – positive postconditions only – are a natural and useful abstraction of attacks (Templeton and Levitt 2000; Ammann, Wijesekera, and Kaushik 2002). Given this, the "attack graph" became what the AI planning community calls a *relaxed planning graph*.[5] Indeed, Amman et al. (2002) reinvented relaxed planning graph generation, as well as relaxed plan extraction, as in the FF system (Hoffmann and Nebel 2001). The idea of monotonic attacks was widely adopted (e. g. (Jajodia, Noel, and O'Berry 2005; Ou, Boyer, and McQueen 2006; Noel et al. 2009; Ghosh and Ghosh 2009)). Many works can, from an AI planning perspective, be understood as variants of, and analyses on, relaxed planning graphs in a practical security modeling and support context.

A close relative of attack graphs are *attack trees* (e. g. (Schneier 1999; Mauw and Oostdijk 2005; Sarraute, Richarte, and Obes 2011)). Indeed the literature does not clearly distinguish these terms and, like "attack graph", "attack tree" is heavily overloaded. In a nutshell, attack trees arose from early attack graph variants meaning the action descriptions themselves. This developed into what has recently been coined "Graphical Security Models" (Kordy et al. 2013): Directed acyclic AND/OR graphs organizing known possible attacks into a top-down refinement hierarchy over attack actions and sub-actions, annotated with costs, success probabilities, etc. The intention is for the user to write this hierarchy, and for the computer support to analyze how costs and probabilities propagate through the hierarchy.[6] In comparison to attack graphs and planning formulations, this has computational (and potentially other) advantages, but cannot find unexpected attacks, arising from unforeseen combinations of atomic actions.

The reader may wonder whether simulated pentesting may be viewed as a special case of model checking. To the extent that model checking safety properties is "the same as" planning, that is true. A major difference to the usual focus in model checking is that verifying correctness is of limited relevance. We cannot conclude "proved safety" if no attack exists according to the model. The model is intended to capture potential vulnerabilities at a high level, not to exactly reflect the system under scrutiny, nor an over-approximation thereof. Neither would be feasible given the overwhelming size and dynamics of modern computer networks.

Finally, a few words are in order regarding game theory. *Why model only the attacker, not also the defender?* The author is far from wishing to claim a conclusive answer to that question. Our answer here basically is "in order to focus on simpler things first". That said, model accuracy may be an issue. It is already very difficult to come up with a useful model of the attacker's point of view, never mind futures of interaction with a defender. It is also not clear that a single network intrusion (an "attack") is naturally viewed as a game. After all, once the defender actually notices an ongoing intrusion, the "game" is basically over. Pentesting is traditionally viewed as a means to strengthen the defenses, not as an interleaved interaction with attackers.[7]



	Explicit Network Graph	Monotonic Actions	General Actions
States	PO−CHP (i) (iii) –– (viii) CoreSec−POMDP	Attack−Asset POMDP (iii) (iv) (vi) [(i) (vii) (viii)]	Factored POMDP [(i) (iii) (vii) (viii)] Current POMDP Model (Sarraute et al. 2012)
Action Outcomes	Canadian Hacker Problem (CHP) (i) (iii) –– (viii) CoreSec−MDP	Attack−Asset MDP (iii) (iv) (vi) (viii) [(i) (vii)] (Durkota and Lisy 2014)	Factored MDP [(i) (iii) (vii) (viii)]
None	Graph Distance (i) –– (v) CoreSec−Classical (Lucangeli et al. 2010)	Delete−Relaxed Classical Planning (i) –– (iv) Attack Graphs e.g. (Amman et al. 2002)	Classical Planning (i) (ii) [(iii)] CyberSecurity (Boddy et al. 2005)

Figure 4: Our model taxonomy, overview. Within each class, the underlying formalism name is given at the top (known formalisms blue, new formalisms red). The underlying assumptions are given in the middle; recommended (but not strictly necessary) ones are put into square brackets "[]". A corresponding concrete pentesting-related model, if any such model exists, is named at the bottom.

A Model Taxonomy

Coming back to the AI sequential decision making perspective, we now systematize our observations in the form of a model taxonomy, multiplying the most relevant distinction lines on dimensions (A) and (B). This serves to provide an overview, and to identify new interesting formalism special cases. We briefly analyze these special cases, and discuss some relations to known problems. Consider Figure 4.

Dimension (A), y-axis, simply corresponds to our three alternate models of uncertainty. In dimension (B), x-axis, the leftmost class pertains to explicit network graph models, where each action corresponds to an edge between a pair of hosts, and the goal is to reach a single target host. The middle class pertains to monotonic actions in the sense of delete-relaxed STRIPS. The rightmost class pertains to general action descriptions, not making that restriction.

Assumption (i) known network graph is recommended given the complexity of modeling and reasoning about network graph topology probabilities. Assumptions (iii) static networks and (vii) configuration-deterministic actions are recommended as they are benign, i. e., uncritical in this application. Monotonic actions necessarily assume (iii) as

[5] This observation was made earlier on by Lucangeli et al. (2010). Boddy et al. (2005) also mention it, though not as clearly.

[6] Many attack tree models are equivalent to AI *formula evaluation* (e. g. (Greiner 1991; Greiner et al. 2006)). This appears to remain unnoticed by both communities. To our knowledge, the only authors pointing it out are Lisý and Píbil (2013).

[7] Note the differences to the GameSec workshop series, and security games (e. g. (Tambe 2011)) concerned with physical infrastructures and defenses: There are many useful applications of game theory in security at large. The question is whether such applications exist for network security simulated pentesting in particular.

changes to the network would naturally include effects overwriting the previous configuration. Recall that (iii) and (vii) together imply that repeating the same action is not useful, entailing the apply-once constraint (viii). Regarding the action outcome uncertainty models, recall that under assumptions (i), (iii), and (vii), and especially when (vi) is also made, outcome probabilities can be obtained from simple statistics running exploits against hosts with particular observable configuration parameters.

The bottom row of deterministic models, and the rightmost column of general action models, are standard formalisms (actually, classes of standard formalism variants) so there is not much need for discussion. Matters are more interesting for the new classes, to the top left.

Let's consider first the case of an explicit network graph model with uncertainty over action outcomes. CoreSec-MDP is a concrete model from this class. Formulated more generically, we are facing an MDP whose states are subsets of compromised network graph nodes, and whose probabilistic transitions correspond to network graph edges. We baptize this the *Canadian Hacker Problem (CHP)* here because it is reminiscent of the stochastic *Canadian Traveller Problem (CTP)* (e. g. (Papadimitriou and Yannakakis 1991; Eyerich, Keller, and Helmert 2010)). Each exploit a pertains to a hop from host ?s to ?t. Due to the apply-once constraint, the "edge" a may be attempted at most once, so we can perceive the edge as "blocked" with probability $1-p(a)$. Hence the input to CHP can be viewed as a graph G with initial and goal hosts, where edges have weights and blocking probabilities. There are two differences to common formulations of CPT. First, in order to "learn" whether or not an edge is blocked, we need to execute the move, i. e., attempt the corresponding exploit (instead of only visiting the edge's start node). Second, whereas in the CTP at any point in time we "are at" one unique graph node (location), in CHP at any point in time we "have" *a subset of graph nodes* (compromised hosts). Therefore, there is no need to "drive back" when an edge is found to be blocked during execution.

Stochastic CTP is usually formulated as a deterministic POMDP; one could use such a formulation for CHP as well by considering edge blocking information a partially observable part of the state. We find the MDP formulation preferable as it makes the trivial nature of the "sensing" here more explicit. This changes for the CoreSec-POMDP model, which corresponds to a deterministic-POMDP generalization of CHP that we baptize *Partially Observable CHP (PO-CHP)*. Instead of just the compromised nodes, we now also have (static) configuration predicates. Instead of an explicit blocking probability, each graph edge has a conjunctive precondition over the configuration predicates, and we are given a probability distribution (the initial belief b_0) over these predicates. In addition to the graph edge moves (the exploits) themselves, we have sensing actions providing partial information about that probability distribution. Overall, this can be viewed as a variant of stochastic CTP, with sensing actions, no need to "drive back", and a complex representation of inter-dependent blocking probabilities.

Attack-Asset MDPs form a sub-class of goal-directed factored MDPs arising as a minimal extension of delete-relaxed STRIPS. The state variables P (a finite set of Boolean propositions), initial state $s_0 \subseteq P$, and goal $G \subseteq P$ are exactly as in STRIPS. The action descriptions a are as in delete-relaxed STRIPS with action costs, except that, in addition to the precondition $pre(a) \subseteq P$, add list $add(a) \subseteq P$, and non-negative action cost $c(a)$, we also have the success probability $p(a)$. Each action a has two possible outcomes, success with probability $p(a)$ where $add(a)$ becomes true, or failure with probability $1-p(a)$ where the effect is empty. The available actions are part of the state, and when a is applied it is not available anymore afterwards. There is, to the author's knowledge, no published work on this problem class except for the aforementioned STAIRS paper by Durkota and Lisý (2014). They give a very similar syntax (calling it an "attack graph"), and outline a few first algorithm ideas for finite-horizon expected reward optimization.

Attack-Asset POMDPs, finally, are the generalization of CoreSec-POMDP which allows exploits to have arbitrary conjunctive non-static preconditions, and arbitrary conjunctive positive effects in case of success. Viewed relative to Attack-Asset MDPs, this means that the state variables, initial state, goal, and exploit-action descriptions remain the same. The state variables are no longer fully observable. Exploit actions succeed if their precondition holds, returning observation "succeed", and fail if their precondition does not hold, returning observation "fail". In addition, deterministic sensing actions are introduced; these may, e. g., be described in terms of conditional observations as in Figure 2.

A third dimension, not represented in Figure 4, is of course the optimization criterion: *What is the attacker trying to achieve?* The options for answering this question, at least those we have considered so far, are all well known. Which options make sense depends on our position in the taxonomy, and on additional assumptions. So this "third dimension" is not fully orthogonal to the other ones.

For the deterministic models, multi-criteria optimization (separating execution time from success statistic) may be interesting. Simple criteria we could apply to all probabilistic models result from annotating rewards (to the goal, or in more complex forms), and maximizing either finite-horizon expected reward or cumulative discounted expected reward. Both are valid options. Notable issues are that simulated pentesting is not an online problem (we cannot actually execute the attack, we are analyzing the threat of potential attacks), and a finite horizon makes little application sense (unless the horizon is so large as to encompass all finite runs, cf. below). As each intrusion happens within a comparatively small time frame, discounting is unintuitive. Finally, it is unclear how human users would set the "rewards", and the outcome of the simulated pentest will crucially depend on this setting. Goal-directed formulations might be easier to handle, requiring only to specify the target hosts/assets. Indeed, this is commercial practice with CoreSec-classical.

One natural option is to minimize non-discounted expected cost-to-goal, in a Stochastic Shortest Path (SSP) formulation. Goal states are absorbing. All our problem classes may contain dead-ends, which reflects the fact that, under highly adverse circumstances, the attacker cannot reach the target. But a real-world attacker, of course, always has the option to give up. We can include such an action into any of our models, leading to an absorbing state with an empty precondition but at a high cost. In CHP, PO-CHP, and Attack-Asset (PO)MDPs, every run is finite due to the (implicit or explicit) apply-once constraint. So, when including a give-up action, every run reaches an absorbing state in finite time, and we have an SSP (in fact, a simple special case thereof). For more general (PO)MDP models, to ensure improper policies incur infinite cost, it suffices to assume strictly positive action costs, a very reasonable assumption in pentesting.

Expected cost-to-goal also has disadvantages. Setting the give-up cost is akin to setting the goal reward, incurring the same modeling difficulty. While this could potentially be avoided by different handling of dead-ends (e. g. (Kolobov, Mausam, and Weld 2012; Teichteil-Königsbuch 2012)), it is not clear anyway that expected cost-to-goal – "What is the minimal expected cost for an attacker to reach the critical data?" – is the question security teams are most interested in. A more natural approach seems to characterize different kinds of attackers through different *cost budgets*, and to *maximize goal probability* given such a budget.

This optimization criterion is, again, known. Kolobov et al. (2011) define a very general class MAXPROB of MDPs based on this criterion, and design algorithms suitable for solving these. Our case here is simpler in that we can assume to be facing, again, a finite-runs SSP. We maximize non-discounted reward where reaching the goal gives reward 1, and all other rewards are 0. The remaining budget b is part of the state, and an action is applicable only if its cost does not exceed b. Goal states and states with no applicable actions are absorbing. In CHP, PO-CHP, and Attack-Asset (PO)MDPs, every run ends in an absorbing state after a finite number of steps, simply by the apply-once constraint. In general, the same is true when assuming strictly positive action costs, as we will eventually run out of budget.

Research Challenges

Many promising research opportunities are given by the new formalism special cases in Figure 4. These are substantially more restricted than general Factored (PO)MDP models. At the same time, these restricted sub-classes have a clear practical motivation. *Can we exploit the restrictions for designing effective algorithms and, with that, practically feasible yet more accurate simulated pentesting tools?*

To say a few words on Attack-Asset MDPs in particular: They are extremely close to classical planning, yet already a rich pentesting model compared to those in practical use, and straightforwardly obtained from these latter models by using success statistics as probabilities instead of a source of action costs. Consider determinization techniques, justly criticized for trivializing probabilistic problems by ignoring the intricacies of multiple action outcomes. Well, we got only two outcomes now, of which one is "nothing happens".[8] In the all-outcomes determinization, we can omit the (empty) failed outcomes, so every probabilistic action yields a single deterministic action. Even better, these deterministic actions have no delete effects! Determinized plans and inadmissible determinization heuristics correspond to standard relaxed plan heuristics. We can compute lower bounds on expected cost through admissible approximations of h^+. A state is a dead end iff there is no determinized plan. Defining a "landmark" to be a set of action outcomes of which any goal-reaching MDP execution must use at least one, the landmarks are exactly the disjunctive delete-relaxation action landmarks of the determinization.

Our model systematization misses non-deterministic planning, which could serve as a trade-off in between the classical and MDP models. Interesting questions also arise regarding the relationship between different model subclasses. For example, are there feasible ways of making the MDP approximation of the POMDP model more accurate? One could maintain partial configuration information, like deleting a flag in case an exploit relying on a particular configuration property fails, thereby excluding other exploits relying on the same property. Can we design this kind of method in a way giving a guarantee – optimism, pessimism, or an error bound – on the success probability relative to the POMDP after the same action outcome sequence?

Given its purpose of guiding human security officers, simulated pentesting comes with a variety of challenges beyond finding single attack plans. As reflected widely in the attack graph literature, we should provide a *collection* of attack plans, and these should be *diverse* so as to cover a range of potentially dangerous intrusions. Plan diversity has been considered, but has not received extensive attention, and the specifics of the pentesting application constitute new challenges and opportunities. There are possible connections to software testing, as system/vulnerabilities coverage may be a suitable meta-criterion for the pentesting process as a whole. What security officers ultimately want is a *situation report*, overviewing the most relevant past and current threats, along with the past and possible counter-measures.

In summary, simulated penetration testing challenges AI sequential decision making at all levels, from classical planning, via Canadian Traveller problems and MDPs, to factored POMDPs. The challenge regards both algorithms and models, the former especially concerning the exploitation of special-case problem structure, the latter especially concerning the identification of formalism sub-classes, as well as the design of practical and accurate probability distributions. A challenge not even touched in the present paper yet are socio-technical attacks, including things like reconnaissance in social networks, whose simulation poses formidable challenges even when disregarding the unrealistic (and ethically questionable) prospect of releasing Turing Test chatbots into Facebook. At large, the author believes that simulated pentesting has the potential to become a major application and research challenge for AI sequential decision making.

Acknowledgments. I thank the ICAPS'15 Chairs for inviting me to write this paper. Thanks to Chris Beck, Ronen Brafman, Olivier Buffet, Carmel Domshlak, and Sheila McIlraith for feedback, and especially to Olivier Buffet and Carlos Sarraute for their collaboration on this topic.

[8] More precisely, nothing happens except we cannot use the same action again, which is why we are not, although almost, in what Keyder and Geffner (2008) call "self-loop MDPs".

References

Ammann, P.; Wijesekera, D.; and Kaushik, S. 2002. Scalable, graph-based network vulnerability analysis. In *ACM Conf. Computer and Communications Security*.

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of action generation for cyber security using classical planning. In *ICAPS'05*.

Burns et al. 2007. *Security Power Tools*. O'Reilly Media.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *AI* 69(1–2):165–204.

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

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-quality policies for the Canadian traveler's problem. In *AAAI'10*.

Ghosh, N., and Ghosh, S. K. 2009. An intelligent technique for generating minimal attack graph. In *SecArt'09*.

Greiner, R.; Hayward, R.; Jankowska, M.; and Molloy, M. 2006. Finding optimal satisficing strategies for and-or trees. *AI* 170(1):19–58.

Greiner, R. 1991. Finding optimal derivation strategies in redundant knowledge bases. *AI* 50(1):95–115.

Hansen, E., and Feng, Z. 2000. Dynamic programming for POMDPs using a factored state representation. In *AIPS'00*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR* 20:291–341.

Huber, M.; Kowalski, S.; Nohlberg, M.; and Tjoa, S. 2009. Towards automating social engineering using social networking sites. In *International Conference on Computational Science and Engineering (CSE'09)*.

Jagatic, T. N.; Johnson, N. A.; Jakobsson, M.; and Menczer, F. 2007. Social phishing. *C. ACM* 50(10):94–100.

Jajodia, S.; Noel, S.; and O'Berry, B. 2005. Topological analysis of network attack vulnerability. In *Managing Cyber Threats: Issues, Approaches and Challenges*. chapter 5.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *AI* 101(1–2):99–134.

Keyder, E., and Geffner, H. 2008. The HMDP planner for planning with probabilities. In *IPC 2008 planner abstracts*.

Keyder, E., and Geffner, H. 2009. Trees of shortest paths vs. Steiner trees: Understanding and improving delete relaxation heuristics. In *IJCAI'09*.

Kolobov, A.; Mausam; Weld, D. S.; and Geffner, H. 2011. Heuristic search for generalized stochastic shortest path MDPs. In *ICAPS'11*.

Kolobov, A.; Mausam; and Weld, D. S. 2012. A theory of goal-oriented MDPs with dead ends. In *UAI'12*.

Kordy, B.; Kordy, P.; Mauw, S.; and Schweitzer, P. 2013. ADTool: security analysis with attack-defense trees. In *International Conference on Quantitative Evaluation of Systems (QEST'13)*.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems IV*.

Lippmann, R., and Ingols, K. 2005. An annotated review of past papers on attack graphs. Technical report, MIT.

Lisý, V., and Píbil, R. 2013. Computing optimal attack strategies using unconstrained influence diagrams. In *Pacific Asia Workshop on Intelligence and Security Informatics*.

Lucangeli, J.; Sarraute, C.; and Richarte, G. 2010. Attack planning in the real world. In *SecArt'10*.

Mauw, S., and Oostdijk, M. 2005. Foundations of attack trees. In *International Conference on Information Security and Cryptology (ICISC'05)*.

Monahan, G. 1982. A survey of partially observable Markov decision processes. *Management Science* 28:1–16.

Noel, S.; Elder, M.; Jajodia, S.; Kalapa, P.; O'Hare, S.; and Prole, K. 2009. Advances in topological vulnerability analysis. In *Cybersecurity Applications & Technology Conference for Homeland Security (CATCH'09)*.

Ou, X.; Boyer, W. F.; and McQueen, M. A. 2006. A scalable approach to attack graph generation. In *ACM Conference on Computer and Communications Security (CCS'06)*.

Papadimitriou, C. H., and Yannakakis, M. 1991. Shortest paths without a map. *TCS* 84:127–150.

Phillips, C., and Swiler, L. P. 1998. A graph-based system for network-vulnerability analysis. In *New Security Paradigms Workshop*.

Ritchey, R. W., and Ammann, P. 2000. Using model checking to analyze network vulnerabilities. In *IEEE Symposium on Security and Privacy*.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2011. Penetration testing == POMDP solving? In *SecArt'11*.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2012. POMDPs make better hackers: Accounting for uncertainty in penetration testing. In *AAAI'12*.

Sarraute, C.; Richarte, G.; and Obes, J. L. 2011. An algorithm to find optimal attack paths in nondeterministic scenarios. In *Workshop on Security and AI*.

Schneier, B. 1999. Attack trees. *Dr. Dobbs Journal*.

Sheyner, O.; Haines, J. W.; Jha, S.; Lippmann, R.; and Wing, J. M. 2002. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*.

Tambe, M. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

Teichteil-Königsbuch, F. 2012. Stochastic safest and shortest path problems. In *AAAI'12*.

Templeton, S. J., and Levitt, K. E. 2000. A requires/provides model for computer attacks. In *Workshop on New Security Paradigms*.