Knowledge Reformulation and Deception as a Defense Against Automated Cyber Adversaries

Ron Alford
The MITRE Corporation
McLean, VA
ralford@mitre.org

Lukáš Chrpa
Czech Technical U. in Prague
Prague, Czech Republic
chrpaluk@fel.cvut.cz

Mauro Vallati
U. of Huddersfield
Huddersfield, UK
m.vallati@hud.ac.uk

Andy Applebaum
The MITRE Corporation*
McLean, VA
applebau@ucdavis.edu

Abstract

Leveraging automated planning has been shown to be advantageous for automating network penetration testing, providing a foundation to generate intelligent approaches to attacking a target system. Unfortunately, this same technology has the potential to be abused by actual attackers, presenting a challenge to defenders. In this paper, we investigate how we can leverage ideas from the deception community to reduce the automated planning capacity of an actual attacker. Our extensive experimental analysis sheds some light on the susceptibility of planning-based attackers to knowledge modifications, potentially yielding to new insights on future techniques for cyber defense.

Introduction

Penetration testing, along with its sister fields of red teaming and adversary emulation, remains a staple for organizations looking to enhance their cybersecurity posture: by executing a controlled attack against your own system, you can identify and remediate gaps in your defenses before an actual adversary exploits them. Traditionally executed manually, in more recent years the community has started to shift towards more *automated* solutions; as it stands, a manual test is time consuming and personnel dependent, whereas an automated test can be quick, easily repeatable, and requires less oversight. Among these automated testing approaches, those that use automated planning have proven both adaptable and effective (Hoffmann 2015; Applebaum et al. 2016), presenting as more intelligent and dynamic than scripted or fixed approaches. Unfortunately, these same automated planning technologies can be used for both controlled penetration tests as well as *actual* adversarial incursions; in doing so, an adversary could execute attacks quicker and more stealthily, bypassing many of today's modern defense-in-depth protections. It is not expressly clear how we should defend against such an automated planning-enabled adversary.

Automated planning approaches rely on models encoding knowledge about actions and the effect these actions have on a state of the world. This knowledge is to be used by a planning engine to generate plans, i.e. sequences of actions that allow to reach a goal state given a predefined initial state. The importance of such knowledge models for automated planning approaches has been well argued by McCluskey, Vaquero, and Vallati (2017), as well as the robustness of state-of-the-art planning engines with regards to poorly engineered knowledge models (Vallati and Chrpa 2019).

In this work, we seek to answer the following question: *How can a network be designed to reduce the automated planning capacity of a prospective automated attacker?* We are interested in changes to the real-world that can result in knowledge models that give a strategic advantage to the "blue" (i.e., defensive) team, taking inspiration from the existing "cyber deception" community (Ferguson-Walter et al. 2021); while we argue for changes to actual systems, such changes need not be made explicitly in the real-world, but rather the adversary need merely to perceive them. Differing from existing work, where the knowledge models were modified agnostic to the corresponding application scenario, here we investigate if domain-specific expertise can also be leveraged to reduce an attacker's effectiveness.

This paper addresses the above question systematically, by defining a set of hypotheses to be tested considering a wide range of modifications that can be performed on a Windows enterprise network. Our extensive experimental analysis shows that even minor modifications, such as adding a fictional disconnected network or unusable users, can have a dramatic impact on the planning abilities of an attacker – and therefore can provide a remarkable strategic advantage to the blue team. Besides being an interesting exercise for testing the robustness of planning engines in a real-world application scenario, our work lays the foundation for the future of automated cybersecurity, where both red and blue team exploit automated techniques to counteract each other.

Background

Cybersecurity and Deception Cybersecurity is a fertile ground for artificial intelligence (AI) research, posing challenges across many aspects of automated perception and decision making. Indeed, cybersecurity has long been a target domain for automated planning, with early work including attack graphs (Sheyner et al. 2002) and defensive course of action generation (Boddy et al. 2005), and more recently on use for automated penetration testing (Hoffmann 2015).

Cyber deception – wherein a defender plants false infor-

*Work performed while at The MITRE Corporation.

mation on a network to throw off adversaries – is a key component of network defense, and its development is a burgeoning field of research. Deceptions include network-level emplacements, such as honeypot hosts (Provos and others 2004) and network tarpits (Hunter, Terry, and Judge 2003), and host-level deceptions, such as fake user credentials (Herley and Florêncio 2008) and generated content (Chakraborty et al. 2019). Automated design of deception operations requires defenders to choose an objective. Some design mechanisms are oblivious to the particulars of an adversary, such as obscuring the structure of a network for all hosts (Achleitner et al. 2017), or blending deceptive services into an environment so that they match surrounding services (Fraunholz and Zimmermann 2017). For tasks with a fixed adversary, such as in malware analysis and explosion, toolkits such as Dodgetron (Sajid et al. 2020) can leverage static analysis and repeated experimentation. For learning adversaries, Walter, Ferguson-Walter, and Ridley (2021) show the effectiveness of deception on the default reinforcement learning strategies of CyberBattleSim[1] high-level simulation environment.

Automated Planning Automated planning is a field of AI that deals with finding a sequence of actions transforming the environment from an initial state to a desired goal state (Ghallab, Nau, and Traverso 2004). Classical planning considers static and fully observable environments, as well as deterministic and instantaneous action effects. The environment is described by first-order logic *predicates*. *States* are defined as sets of *atoms* that are grounded predicates whose variable symbols are substituted by constants – problem-specific objects. An *action* is specified via its unique action name, variable symbols (parameters) appearing in the action, a formula over predicates representing its precondition and a formula over predicates representing its effects. Action effects, roughly speaking, represent how the state of the environment will change after the action is applied. Grounded instances of an action are obtained by substituting action's parameters for constants – problem-specific objects. A grounded action is *applicable* in a given state if and only if the precondition of the action holds in that state. Application of a grounded action in a given state (if possible) results in a state where the effects of the action becomes true.

In classical planning, a planning task is composed by two models: a *domain model* is specified via sets of predicates and actions. A *problem instance* is specified via objects (that are substituted for free variables in predicates and actions), an initial state, and a set of goal atoms. A *solution plan* for a planning task is a sequence of grounded actions such that its consecutive application (starting in the initial state) results in a state in which all the goal atoms are true.

Considered Cybersecurity Scenario

We focus our analysis on the well-known cybersecurity subproblem *CALDERA* (Miller et al. 2018),[2] used as a benchmark in the 2018 International Planning Competition (IPC). The domain model is encoded in PDDL, and features a representation of an adversary looking to compromise a Windows enterprise network by leveraging credential re-use to laterally move and propagate to each of the hosts in the network. This domain is both simple and realistic: the encoded tradecraft of credential re-use is used frequently by adversaries in the wild, whereas typical cybersecurity planning domain models by contrast focus on exploitation, which is only a small piece of most adversary toolkits. Additionally, the CALDERA model is built from open-source software[3] that can run fully automated red team exercises, helping to establish a level of realism.

The CALDERA model also offers two other interesting nuances. First, it encodes a mixture of "discovery" and "acting," where the planning engine needs to explicitly discover an object before it is able to use it. This adds realism, as an adversary never walks into a network knowing what that network consists of and needs to discover it through the course of compromising it. Second, the domain model is monotonic and without dead-ends; a planning engine could eventually achieve all objectives simply by making random action choices at any given time. While this seems unrealistic, indeed it is the case that for the given tradecraft that there are no necessarily "wrong" decisions.

The CALDERA domain model features eight actions. Each action has an analog to real adversary behavior, and is intended to model the real constraints of an adversary executing the behavior in the wild. The actions each take a variable amount of parameters, ranging from only two for the simplest ("Get Computers") to 11 for the most complex ("Net Use"). The domain models also features 30 predicates, used to describe the state of the network.

Ransomware Domain Model

As part of this paper we extend the CALDERA domain model to include the case of "ransomware." This entails three changes: adding a predicate to mark that a host is ransomed, adding a precondition of "not ransomed" to existing actions, and adding a new action to allow the adversary to ransom a host. This new action requires the adversary to already have a foothold on the host and results in that host being inoperable – i.e., no more actions can be run on it.

These changes add complexity (specifically *dead-ends*) as a planning engine needs to make sure that it does not ransom a host before using it to propagate to the next host. The presence of dead-ends means that there are areas of the search space that do not have a path to reach a state where the goal conditions are satisfied, which can make the search process particularly challenging.

Analysis

For the sake of clarity, the analysis is designed around eight hypotheses, divided into two classes. The first class, domain-independent hypotheses, considers the perspective of an automated planning expert with no knowledge about the specific application domain; this class gives us the opportunity to investigate some general techniques that are agnostic with regards to CALDERA and cybersecurity. The second class

[1] https://github.com/microsoft/cyberbattlesim

[2] shorturl.at/yCIQU

[3] https://caldera.mitre.org

gives instead the point of view of an expert of cybersecurity, with limited knowledge of automated planning. This second class of hypotheses mostly leverages the characteristics of the cybersecurity scenario to negatively affect the performance of planning engines. Due to the different natures of the perspectives, some hypotheses can be conflicting.

Domain-independent Hypotheses

Hypothesis 1 *Adding objects that lead to a larger grounded representation slows down the planning process and increases memory requirements.*

Hypothesis 2 *Adding predicates that increase the branching factor, particularly by forcing the planning engine to take into account additional actions, increases memory requirements and slows down the planning process.*

Hypothesis 3 *Adding predicates that introduce dead-end states, or that introduce paths disjoint from the goal, slows down the planning process as planning engines might get trapped in a "dead-end part" of the search space.*

Domain-specific Hypotheses

Hypothesis 4 *Adding random stand-alone network objects does not slow down the planning process significantly, as the additions are irrelevant to reach the goal.*

Hypothesis 5 *Adding additional information immediately accessible from the start host will cause a greater slowdown than adding that same information to a randomly selected host, as adding to the start host will require the planning engine to interact with the information.*

Hypothesis 6 *Adding a network completely separated from the original problem will significantly slow down the planning process as planning engines might be tempted to explore the fake part of the network.*

Hypothesis 7 *Adding a network with some overlap with the real one slows down the planning process, as the engine has to figure out if there are some ways to exploit the overlap.*

Hypothesis 8 *Adding credential overlap, i.e., by making more users, admins or caching additional credentials, will speed up the planning process as this makes it easier to laterally move. Conversely, adding users in a way that avoids credential overlap will slow the planning process down, as there is more user accounts to investigate.*

Experimental Approach

In our analysis, we use the following workflow: (i) Identify a set of problems to focus on; (ii) Run each planning engine on the original problems; (iii) For each deception scenario, apply the needed modifications to the problems, and run the planning engines 25 times to account for stochasticity; (iv) Compare the results for each planning engine under each deception scenario against that engine's performance in the unmodified case. In following this process, our aim is to provide empirical results backing up our hypotheses.

We focus our analysis on three problems, having a version of each for both the CALDERA and the ransomware domain. All problems feature one Windows domain and an initial state of the adversary having a foothold on one of the hosts; for the CALDERA model, the goal is to compromise all hosts, and for the ransomware model the goal is to ransom

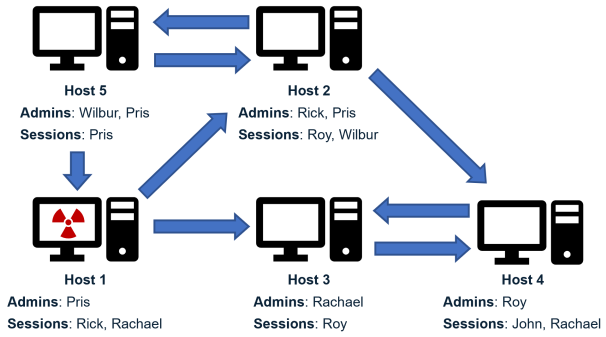

Figure 1: The network for the Manual problem. A blue arrow from host A to host B implies that a foothold on A will enable lateral movement to B.

all hosts. The first two problems are taken from the IPC submission: **Problem 3** consists of two hosts and two users and is intended to be "easy," while **Problem 8** consists of three hosts and six users and is slightly more complex, as each host only has one admin and one user account with cached credentials, and each user is an admin on at most one host.

We also introduce a new problem not part of the IPC benchmark, which we refer to as the **Manual** problem, visualized in Figure 1. This problem features five hosts and six user accounts, all part of the same Windows domain. The initial state features the adversary having a compromise on Host 1, with the goal of compromising Hosts 2 through 5 in the CALDERA model, and ransoming all hosts in the ransomware one. The problem is slightly more complex than the other two, offering different paths and more variability.

Deception Scenarios

Our goal is to present to a set of planning engines a set of different network scenarios, with the idea of tripping up the engine before it even gets into the acting stage of attacking the environment. In practice, the changes that we explore could be implemented as deceptions or as actual network modifications. Our scenarios furthermore follow two main guidelines: (1) we try to make as few changes as possible to the original problem and the optimal solution plans, and (2) we focus on modifications that can be realistically implemented within a Windows environment. We ultimately designed 26 different scenarios each fitting into one of five categories, summarized in Table 1.

Adding Random Objects. In these scenarios, we add a random number of objects disconnected from the original problem. This type consists of six scenarios. *random-objects* will add an assortment of objects, whereas *random-defined-objects* will add an assortment of objects and also *properties* for said objects. As an example, adding a "user" object in the *random-objects* case simply adds the user, but in the *random-defined-objects* case adds a user and assigns it a string username. This type of modification tests two hypotheses that are somewhat at odds: we would expect Hypothesis 1 to suggest a slowdown due to general planner properties, but Hypothesis 4, from a domain-specific perspective, would suggest no change as the planning engine would not consider such additions as relevant.

Name	1	2	3	4	5	6	7	8	Description
random-objects	X			X					Add random objects without properties.
random-defined-objects	X			X					Add random objects with properties., but no connection to other objects.
random-defined-cred	X			X					Add credentials with properties but no connection to other objects.
random-defined-domain	X			X					Add domains with properties but no connections to other objects.
random-defined-host	X			X					Add new hosts with properties but no connections to other objects.
random-defined-user	X			X					Add users with properties but no connections to other objects.
first-noncached-admin-user					X			X	Add a new user as admin on the start host.
random-noncached-admin-user					X			X	Add a new user as admin on a random host.
random-new-user								X	Add a new non-admin user and do not cache it.
first-cached-nonadmin-user					X			X	Add a new non-admin user and cache it on the first host.
random-cached-nonadmin-user					X			X	Add a new non-admin user and cache it on a random host.
new-multicached-user								X	Add a new non-admin user and cache it on multiple random hosts.
multi-new-multicached-user								X	Add multiple non-admin users and cache them on multiple random hosts.
first-dual-homed-host		X			X				Add a new domain and join the start host to it.
random-dual-homed-host		X			X				Add a new domain and join a random existing host to it.
random-dual-homed-user		X							Add a new domain and join a random existing user to it.
dual-homed-multicached-users		X							Add a new domain with new users, cached randomly.
dual-homed-host-and-users		X							Add a new domain with new users and add an existing host to it.
first-cached-credentials			X		X			X	Cache a random user's credentials on the start host.
random-cached-credentials			X		X			X	Cache a random user's credentials on a random host.
first-user-admin			X		X			X	Make a random user admin on the start host.
random-user-admin			X		X			X	Make a random user admin on a random host.
small-disconnected-domain	X	X		X		X			Add an unreachable network with 1 host and its own domain and user.
medium-disconnected-domain	X	X		X		X			Add a network of 3 hosts but unreachable from the original hosts.
first-medium-connected-domain	X	X	X		X		X		Add a network with 3 hosts reachable from the start host.
random-medium-connected-domain	X	X	X		X		X		Add a network with 3 hosts reachable from a random original host.

Table 1: Description of each deception scenario; columns labeled "1-8" denote hypotheses 1-8.

Adding Unusable Users. Here we add new users in a way that makes them "unusable" – i.e., they are an admin on a host but have no active sessions, or have an active session but are not an admin on a host. Within this type, we have seven different scenarios to test, ranging from adding a new user as an admin on the start host to adding multiple new users and storing their credentials on multiple hosts. For some of these scenarios, we offer *placement variation*, where we have one scenario doing something on the start host and another doing something on a randomly select host.

Adding a New Domain. The third type of scenario is to add an additional Windows domain and connect some of the original problem to it, in addition to being connected to the original Windows domain. This type of scenario is perhaps the most difficult to execute in the real world, but is not impossible. In total, we test five different scenarios for this type, two of which differ as to if they modify the start host or a random host; these two both follow the idea of selecting an existing host – start or random – and joining it to a new Windows domain. The other three scenarios follow a similar idea, but now focusing more on users.

Modifying the Topology. These four scenarios include selecting an account at random and storing its credentials on the start or a random host, and also selecting an account at random and making it an admin on either the start or a random host. Unlike the other types of modifications, these all do indeed have the potential to modify the topology and change optimal plans. These scenarios test multiple hypotheses. We note that here Hypothesis 3 and 8 disagree and it might depend on a planning technique and a specific problem instance which one of these hypotheses will prevail.

Adding a Network. Here we focus on adding a new network – including at least one domain, host, user, and credential, all connected to each other – generating four scenarios in total. In the simplest, *small-disconnected-domain*, we only generate one host (disconnected from the main network) with one user being admin and having a session on that host. Slightly more complex is *medium-disconnected-domain*, where we add three hosts (disconnected from the main network) and three users, where one of the hosts has a session for an admin on the other; and that host has a session for an admin on the third. The last two scenarios tweak this *medium* idea slightly, by having one of the users being added have a session on either the start host (*first*) or a randomly selected host (*random*). Between these four scenarios we test nearly every hypothesis.

Experimental Settings

For each scenario of the presented planning problems, a time limit of 900 seconds and a memory limit of 4 GB is applied for a planning engine to generate a solution. All the experiments were conducted on Intel Xeon 2.10 GHz.[4]

We selected 5 state-of-the-art planning engines, namely: *FF* (Hoffmann and Nebel 2001), *LAMA* (Richter and Westphal 2010), *Probe* (Lipovetzky et al. 2014), *FDSS 2018* (Seipp and Röger 2018) and *Dual BFWS* (Lipovetzky et al. 2018). We aimed at including planning engines that achieved good performance in IPCs while exploiting very different planning techniques, to better capture the impact of the modifications on a wide and diverse range of approaches.

Results are presented in terms of coverage, i.e., number

[4]Benchmark data are available at `https://github.com/lchrpa/CyberDeception.git`

Type	Scenario	FF		LAMA		Probe		BFWS		FDSS	
		C	P10	C	P10	C	P10	C	P10	C	P10
Control	no-modifications	**75.0**	**2.5**	**75.0**	**39.9**	**75.0**	**15.1**	**75.0**	**4.5**	**75.0**	**18.6**
Adding Random Objects	random-objects	55	2401.5	75	39.9	51	2883.7	50	3001.3	75	18.6
	random-defined-objects	40	4201.8	75	40.0	39	4324.8	39	4322.1	75	18.7
	random-defined-cred	75	2.6	75	39.9	75	14.4	75	4.6	75	18.6
	random-defined-domain	75	2.7	75	39.9	75	14.9	50	3000.7	75	18.6
	random-defined-host	50	3000.6	75	39.9	50	3002.3	50	3001.0	75	18.6
	random-defined-user	75	2.7	75	40.1	75	14.5	50	3000.7	75	18.6
	Average	**61.6**	**1601.9**	**75.0**	**39.9**	**60.8**	**1709.1**	**52.3**	**2721.7**	**75.0**	**18.6**
Adding Unusable Users	first-noncached-admin-user	75	2.8	75	40.0	50	3001.7	50	3000.7	75	18.7
	random-noncached-admin-user	75	2.8	75	39.8	50	3001.7	50	3000.7	75	18.6
	random-new-user	75	2.8	75	39.8	50	3001.6	50	3000.7	75	18.6
	first-cached-nonadmin-user	75	2.8	75	40.7	50	3001.7	50	3000.7	75	19.4
	random-cached-nonadmin-user	75	2.8	75	40.6	50	3001.7	50	3000.7	75	19.3
	new-multicached-user	75	2.8	75	40.5	50	3001.7	50	3000.7	75	19.3
	multi-new-multicached-user	50	3000.6	75	40.7	50	3002.4	50	3001.0	75	19.4
	Average	**71.4**	**431.0**	**75.0**	**40.3**	**50.0**	**3001.7**	**50.0**	**3000.7**	**75.0**	**19.0**
Adding a New Domain	first-dual-homed-host	75	2.7	75	14.5	75	16.7	50	3000.7	75	18.8
	random-dual-homed-host	75	2.7	75	14.1	75	15.0	50	3000.7	75	18.9
	random-dual-homed-user	75	2.7	75	41.1	61	1687.9	50	3000.7	75	19.6
	dual-homed-multicached-users	50	3000.4	75	41.6	50	3001.9	50	3000.8	75	20.3
	dual-homed-host-and-users	50	3000.4	75	15.4	50	3001.9	50	3000.8	75	20.6
	Average	**65.0**	**1201.7**	**75.0**	**25.3**	**62.2**	**1544.6**	**50.0**	**3000.7**	**75.0**	**19.6**
Modifying the Topology	**Average (4 hidden scenarios)**	**75.0**	**2.5**	**75.0**	**27.0**	**75.0**	**15.4**	**75.0**	**4.7**	**75.0**	**19.0**
Adding an Additional Network	**Average (4 hidden scenarios)**	**50.0**	**3002.0**	**75.0**	**40.0**	**50.0**	**3006.1**	**50.0**	**3002.7**	**75.0**	**39.2**

Table 2: (Average) results for the CALDERA problems. C and P10 stand for Coverage and PAR10 score, respectively.

	FF		LAMA		Probe		BFWS		FDSS	
	C	P10	C	P10	C	P10	C	P10	C	P10
no-modifications	**75.0**	**2.9**	**75.0**	**70.7**	**75.0**	**14.6**	**75.0**	**26.9**	**75.0**	**26.4**
Adding Random Objects	**61.8**	**1582.2**	**75.0**	**77.6**	**61.0**	**1689.5**	**52.6**	**2686.5**	**75.0**	**26.4**
Adding Unusable Users	**71.4**	**431.4**	**75.0**	**71.9**	**50.0**	**3001.8**	**50.0**	**3001.8**	**75.0**	**26.8**
Adding a New Domain	**65.0**	**1202.0**	**71.6**	**512.2**	**60.4**	**1759.5**	**50.0**	**3001.8**	**75.0**	**34.7**
Modifying the Topology	**75.0**	**2.9**	**67.7**	**946.1**	**75.0**	**14.6**	**75.0**	**29.7**	**75.0**	**37.2**
Adding an Additional Network	**50.0**	**3002.3**	**75.0**	**110.5**	**50.0**	**3007.2**	**50.0**	**3006.1**	**75.0**	**43.3**

Table 3: Average results for the "ransomware" problems. C and P10 stand for Coverage and PAR10 score, respectively.

of instances solved within the cutoff time, and Penalized average runtime 10, where the average runtime is calculated by considering runs that did not solve the problem as ten times the cutoff time. P10 provides a good tradeoff between runtime and coverage.

Results and Discussion

Table 2 shows the (average) performance achieved by the considered planning engines on the normal CALDERA problems. Results, for each of the scenarios, include all the 25 instances generated for each of the 3 tested problems. The maximum coverage is therefore 75. When there is no significant performance variance within a deception scenario, we only present average results. From the presented results, we can observe that in each category, except *Modifying the Topology*, FF, Probe and BFWS failed to solve some problems because they ran out of memory during the grounding stage. Note that all these planners are built on top of the architecture of the FF planner and hence share the grounding functionality. Interestingly, in problems which these planners managed to solve, the runtime difference to the original encoding was negligible. The naive strategy of adding random objects is not very effective in reducing the planning performance of the considered engines. Similarly, the strategy of adding to the network unusable users shows a very limited detrimental impact. Very interesting are the results produced on the scenario type of adding a new domain: FDSS's performance is unaffected; the performance of FF, Probe and BFWS are quite negatively affected; but LAMA's performance actually benefits from this useless addition. Notably, this scenario type is also one of the most challenging to implement in an actual network. Our tests on modifying the topology show that it does not negatively influence engines' performance. Instead, LAMA's performance actually improves – but that is due to the fact that the modified topology allows the engine to find a shorter path to the goal. Finally, adding an additional network has the most remarkable impact on the performance of the considered engines: for all the considered planning engines except LAMA, performances are reduced. On the one hand, this behavior is highly expected, as the planning engine has a larger search space to explore. On the other hand, in one scenario (small-disconnected-domain) LAMA is again delivering better performance than those on the original instances.

Turning to the ransomware domain model results in Table 3, we have a general remark: this class of problems is more challenging for the considered planning engines, due to the possibility of reaching dead-end states in the search space. The observations made on the results of the normal CALDERA problems, for most of the scenario types, still

hold when the ransomware model is used, and for this reason, we only present averaged results for each deception scenario. In particular, adding random objects and adding unusable users have a limited yet noticeable impact on the planning performance; and adding an additional network still has a generally negative impact on all the planners. Considerable differences in the performance can be found in the adding a new domain and modifying the topology scenario types. Adding a new domain has a negative impact on all of the planners, with LAMA's performance drastically reduced. Most notably, LAMA's and FDSS's performance are also considerably affected by the modifying the topology scenario type problem instances. On this class of problems, LAMA delivers its worst performance, even though in the normal domain this type of modification helped it to perform *better*. Our intuition is that the presence of dead-ends is making these instances extremely challenging for LAMA and FDSS, possibly due to the exploited heuristics.

Summarizing, the analysis provides evidence that *supports* hypotheses 2, 3 (in presence of dead-ends), 6, and 7. In contrast, hypotheses 5, 8 have not been verified. Finally, hypotheses 1 and 4 are neither fully supported nor fully rejected, as the evidence suggests that it strongly depends on the pre-processing technique of the planning engine.

Conclusion

We investigated how to support deception against automated adversaries by exploring the impact that modifications to a Windows enterprise network can have on the planning capabilities of an attacker. A concrete contribution of this work is that we have been able to identify the most promising and suitable strategies that can be put in operation to gain defenders a strategical advantage against a red team exploiting automated planning techniques to define how to attack a target network. Future work will focus on investigating how to automatically lay effective honeypots in a network on the fly.

Acknowledgments

Mauro Vallati was supported by a UKRI Future Leaders Fellowship [grant number MR/T041196/1].

References

Achleitner, S.; La Porta, T. F.; McDaniel, P.; Sugrim, S.; Krishnamurthy, S. V.; and Chadha, R. 2017. Deceiving network reconnaissance using SDN-based virtual topologies. *IEEE TNSM* 14(4):1098–1112.

Applebaum, A.; Miller, D.; Strom, B.; Korban, C.; and Wolf, R. 2016. Intelligent, automated red team emulation. In *Proc. of ACSAC*, 363–373.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proc. of ICAPS*, 12–21.

Chakraborty, T.; Jajodia, S.; Katz, J.; Picariello, A.; Sperli, G.; and Subrahmanian, V. 2019. Forge: A fake online repository generation engine for cyber deception. *IEEE TDSC*.

Ferguson-Walter, K. J.; Major, M. M.; Johnson, C. K.; and Muhleman, D. H. 2021. Examining the efficacy of decoy-based and psychological cyber deception. In *Proc. of the {USENIX} Security Symposium*.

Fraunholz, D., and Zimmermann, M. 2017. Towards deployment strategies for deception systems. *Advances in Science, Technology and Engineering Systems Journal* 2:1272–1279.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Herley, C., and Florêncio, D. 2008. Protecting financial institutions from brute-force attacks. In *IFIP International Information Security Conference*, 681–685.

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

Hoffmann, J. 2015. Simulated penetration testing: From" dijkstra" to" turing test++". In *Proc. of ICAPS*.

Hunter, T.; Terry, P.; and Judge, A. 2003. Distributed tarpitting: Impeding spam across multiple servers. In *Proc. of LISA*, 223–236.

Lipovetzky, N.; Ramirez, M.; Muise, C.; and Geffner, H. 2014. Width and inference based planners: Siw, bfs(f), and probe. In *Proc. of IPC*.

Lipovetzky, N.; Ramirez, M.; Frances, G.; and Geffner, H. 2018. Best-first width search in the ipc2018: Complete, simulated, and polynomial variants. In *Proc. of IPC*.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering knowledge for automated planning: Towards a notion of quality. In *Proc. of K-CAP*, 14:1–14:8.

Miller, D.; Alford, R.; Applebaum, A.; Foster, H.; Little, C.; and Strom, B. 2018. Automated adversary emulation: A case for planning and acting with unknowns. In *ICAPS Workshop on Integrated Planning, Acting and Execution*.

Provos, N., et al. 2004. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, 1–14.

Richter, S., and Westphal, M. 2010. The LAMA planner: guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Sajid, M. S. I.; Wei, J.; Alam, M. R.; Aghaei, E.; and Al-Shaer, E. 2020. DodgeTron: Towards autonomous cyber deception using dynamic hybrid analysis of malware. In *Proc. of the IEEE CNS*, 1–9.

Seipp, J., and Röger, G. 2018. Fast downward stone soup 2018. In *Proc. of IPC*.

Sheyner, O.; Haines, J.; Jha, S.; Lippmann, R.; and Wing, J. M. 2002. Automated generation and analysis of attack graphs. In *Proc. of the IEEE Symposium on Security and Privacy*, 273–284.

Vallati, M., and Chrpa, L. 2019. On the robustness of domain-independent planning engines: The impact of poorly-engineered knowledge. In *Proc. of K-CAP*, 197–204.

Walter, E. C.; Ferguson-Walter, K. J.; and Ridley, A. D. 2021. Incorporating deception into cyberbattlesim for autonomous defense. In *Proc. of the ACD Workshop*.