THESIS

CHARACTERIZING ANTI-FORENSIC ATTACKERS IN CYBERSECURITY DOMAINS WITH STACKELBERG PLANNING

Submitted by

Jason Curcio

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2026

Master's Committee:

Advisor: Sarath Sreedharan

Indrajit Ray

Jeremy Daily

ABSTRACT

CHARACTERIZING ANTI-FORENSIC ATTACKERS IN CYBERSECURITY DOMAINS WITH STACKELBERG PLANNING

The rapid advancement of artificial intelligence has enabled large-scale, automated cyberattacks capable of targeting critical infrastructure with unprecedented speed. Since a perfect defense is often unattainable in complex networks, defenders must strategically force attackers into either objective failure or leaving a detectable footprint. This research addresses this defensive gap by applying Automated Planning to model a self-cleaning adversary within a state-based environment. Utilizing a Stackelberg planning framework, our methodology simulates a game-theoretic dynamic where a defender proactively modifies the environment and the attacker computes an optimal intrusion path in response. This adversarial interaction is evaluated across a simulated, segmented network, ultimately enabling the formal verification of security invariants and providing a framework to strengthen both network architecture and forensic audit trails.

DEDICATION

To my family, friends, and mentors.

TABLE OF CONTENTS

1. Introduction

The explosion of artificial intelligence has fundamentally reshaped the technological landscape, offering unprecedented capabilities in data processing and autonomous decision-making [1]. However, this paradigm shift has introduced a new frontier of risk, particularly within the realm of cybersecurity. As AI-driven tools become more accessible, threats are no longer defined solely by manual, human-speed intrusions. Instead, attackers can launch large-scale automated campaigns capable of targeting critical infrastructure with alarming speed and sophistication [2]. High-profile incidents, such as the Stuxnet worm's interference with industrial control systems [7] or the Colonial Pipeline ransomware attack [11], underscore the devastating real-world consequences of compromised digital systems. In this environment, the struggle between attackers and defenders has evolved into a perpetual game of cat and mouse. As defensive signatures improve, attackers pivot toward novel exploits and anti-forensic techniques, creating an environment where static security measures can be rendered obsolete before they are fully deployed [17].

1.1 The Defensive Gap: Forcing Adversarial Concessions

The current cybersecurity landscape is a patchwork of reactive and proactive measures that often operate in silos. Reactive tools such as Intrusion Detection Systems (IDS), Security Information and Event Management (SIEM), and Endpoint Detection and Response (EDR) focus on minimizing dwell time by triggering alerts during or after an intrusion. Conversely, proactive measures like vulnerability scanners and firewalls aim to prevent breaches entirely. However, these tools often identify isolated vulnerabilities rather than understanding how they can be chained into multi-step attack paths.

The fundamental challenge for a defender is determining how to force strategic concessions from the attacker. In an adversarial environment, a defender's goal is to force the attacker into one of two unfavorable outcomes:

1. Objective Failure: Preventing the attacker from achieving their primary goal (e.g., data exfiltration or system disruption).
2. Detection via Trace: Forcing the attacker to leave a detectable footprint that triggers an immediate and effective response.

Because a perfect defense is often unattainable in complex, modern networks, the goal shifts toward designing systems where trace-free attacks become infeasible. Even the most capable attackers must interact with and modify system logs or system states as they progress through a network.

1.2 Filling the Gap with Automated Planning

This research fills the defensive gap by treating security as a dynamic, state-based problem. By applying automated planning, we can model the strongest feasible attacker within a given environment. Planners can exhaustively search millions of potential system states to discover non-obvious attack vectors that an adversary could exploit.

Using the Planning Domain Definition Language (PDDL), the causal flow of an attack is modeled as such: Action A satisfies the precondition for Action B, which enables Action C. By identifying these causal chains, defenders can strategically break the plan at the most efficient bottleneck. Furthermore, this approach allows for the formal verification of security invariants, proving whether a user without specific privileges can ever reach critical assets. If a planner

finds a solution, the invariant is violated, and the network architecture—like its permission structures and logging mechanisms—must be refined.

1.3 Research Contributions: Characterizing Anti-Forensics

The primary contribution of this research is the automated modeling of an anti-forensics attacker. While existing literature focuses primarily on an attacker's ability to reach a target, this model explicitly accounts for the cleanup phase of the adversarial lifecycle. By incorporating maneuvers intended to erase logs or modify system artifacts into the planning domain, we can characterize the minimum detectable disturbance an attacker must leave behind. This allows for the design of more robust defense strategies that target the attacker's need for invisibility. In doing so, we provide a comprehensive framework for hardening not just the front door of the network, but the audit trails that follow.

2. Background Information

To understand the methodology used in this research, it is necessary to establish the fundamental concepts of Automated Planning. This section provides an overview of Classical Planning, the standard language used to define it, and advanced planning paradigms that account for uncertainty and adversarial interactions.

2.1 Classical Planning

At its core, Classical Planning is the computational study of selecting a sequence of actions to achieve a specific goal. Unlike procedural programming, where a developer explicitly defines how to perform a task, planning is declarative. The problem defines the *what*—the initial state of the world, the available actions, and the desired goal state—and the planner's job is to figure out the *how*.

In a classical planning problem, the environment is assumed to be deterministic, static, and fully observable. Determinism means that every action has a predictable, single outcome. If an agent performs an action, the system transitions to that exact state with 100% certainty. A static environment only changes when the agent acts upon it. There are no external events or natural phenomena changing the state of the world in the background. Finally, full observability means the agent always knows the complete state of the world.

The solution to a classical planning problem is a plan: a linear sequence of actions that transforms the initial state into a state where the goal conditions are met. In the context of cybersecurity, this is analogous to an attacker finding a specific sequence of exploits and movements that leads them from an external entry point to a critical asset.

To standardize how these problems are represented, the research community developed the Planning Domain Definition Language (PDDL). PDDL separates the problem into two distinct files: the domain and the problem. The domain defines the "physics" of the world. It specifies the types of objects and the predicates that describe facts about them. Most importantly, it defines the actions. Each action is composed of parameters, preconditions, and effects. The problem describes a specific instance of the domain. It lists the specific objects in the environment, the exact initial state, and the goal state. As an analogy, if the domain is the rulebook of chess that defines how each piece is allowed to move, the problem is a specific board arrangement where you must find the sequence of moves to reach checkmate.

2.2 Fully Observable Non-Deterministic (FOND) Planning

While classical planning is effective for stable environments, cybersecurity domains often involve unpredictable outcomes. FOND planning addresses this by relaxing the determinism constraint. In FOND, an action can have multiple possible effects, and the agent does not know which will occur until the action is executed. In the context of this research's early iterations, FOND was used to model an environment where the defender's movements were randomized. This forced the attacker to generate a policy instead of a linear plan.

2.3 Stackelberg Planning

Stackelberg planning frames the planning problem using Game Theory, specifically a leader-follower dynamic. This is particularly relevant for modeling defense-offense interactions in cybersecurity.

In a Stackelberg planning task, there are two agents: the leader and the follower. The leader, or defender, moves first. The leader has the power to change the initial state of the world

(e.g., modifying network configurations, altering user permissions, etc.) to make the follower's goal harder or impossible to achieve. The follower, or attacker, moves second. The follower observes the leader's changes and then plans an optimal sequence of actions to maximize their own reward, or minimize their cost, within that new environment.

The goal of Stackelberg planning is to find the optimal move for the leader such that, even if the follower plays perfectly, the damage is minimized or negated. This approach is superior to standard planning for defense because it does not assume a specific attack path. Rather, it optimizes the defense against the worst-case optimal response of the attacker. If the planner cannot find a successful attack plan after the leader's modifications, the network is theoretically verified as secure against that specific threat model.

3. Related Works

3.1 Foundations of Adversary Modeling

Early methodologies, such as Attack Trees [13], provided foundational logic and sequential flow to root goals. High-level attacks could be decomposed into leaf-node exploits, where each exploit satisfies preconditions for subsequent actions. These graphs are highly intuitive, and analysts can perceive threats from any emerged causal relationships. Trees solve small systems very well, but are largely static and fail to account for temporal operations. To address this, Attack Graphs [14] emerged. Unlike trees, graphs represent the transition between system states, thereby mapping the actions an attacker might take to reach their goal. Sheyner's methodology tackled the temporal problem, but both techniques still suffer from the state-space explosion problem. As networks get increasingly large and complex, the number of system states that require search grows exponentially, outpacing the compute required for graph generation. Additionally, they lack the ability to account for active defender participation or environmental changes caused by attackers.

3.2 Automated Planning in Cybersecurity

The field saw a transition from static graphs to automated planning, which meant it shifted towards system-aware modeling. Researchers found the PDDL [9] particularly useful because cyber-actions could be modeled as logical operators with specific preconditions and effects. Boddy et al. [3] were among the first to demonstrate that PDDL could be used for "Course of Action" (CoA) generation, allowing a system to automatically discover multi-step attack paths that a human might overlook. Further advancements by Hoffmann [6] using simulated penetration testing showed that planning could not only find paths, but also evaluate

cost or risk associated with them. Then, more recently, Lucangeli et al. [8] focused on model acquisition—the process of automatically generating these PDDL domains from network scans—highlighting that the efficacy of any planning-based defense is strictly limited by the fidelity of the underlying model.

3.3 Game-Theoretic Defense and Stackelberg planning

The limitation of early mono-agent planning, which failed to capture the adversarial interactivity of cyber warfare, led to the development of game-theoretic formulations known as Stackelberg planning. Speicher et al. [15] pioneered the formalization of this framework by modeling the defender as a "leader" who commits to a strategy, and the attacker as a "follower" who optimizes their path in response. They developed leader-follower search, a bi-level optimization algorithm using branch-and-bound pruning to identify cost-effective defensive strategies that minimize the attacker's maximum utility.

To address the computational bottlenecks inherent in this dual-layer search, Torralba et al. [18] introduced Symbolic Leader Search, utilizing Binary Decision Diagrams to share reachability information across sub-problems and accelerate the generation of optimal defenses. Addressing the scalability issues of applying these models to large networks, Sauer et al. [12] recently proposed Lifted Stackelberg Planning. By reasoning about first-order logic schemas rather than instantiated states, they overcame the grounding bottleneck, enabling the generation of Pareto-optimal mitigation strategies for enterprise-scale infrastructures. Expanding the defensive paradigm from static hardening to active deception, Cates et al. [4] introduced the "Attacker Entrapment Problem." They modeled a hidden defender who covertly manipulates system responses to lure unsuspecting adversaries into trap states (honeypots), offering a dynamic alternative to traditional static defenses.

3.4 Autonomous Cyber Operations and Anti-Forensics

The offensive landscape is undergoing a parallel shift toward autonomous cyber operations, moving from human-driven intrusion to AI agents capable of operating at machine speed. Standen et al. [16] advanced this domain by developing standardized gym environments like CybORG, which allow reinforcement learning agents to be trained against simulated defenders. These environments are critical for enabling agents to autonomously discover vulnerabilities and execute complex attack chains.

A critical, often competing objective in these autonomous missions is anti-forensics: the strategic effort to minimize the forensic footprint of an intrusion. Ghanem et al. [5] highlighted that realistic planning domains must be bi-objective, requiring agents to balance mission success with stealth constraints like artifact wiping and timestamp manipulation. To model the uncertainty inherent in evading detection, researchers have increasingly turned to POMDPs. In this vein, Ren et al. [10] recently demonstrated that LLMs could be integrated into the planning loop, enabling agents to dynamically generate stealth-aware PDDL domains from natural language threat reports and adaptively evade forensic reconstruction.

4. Problem Evolution

Before arriving at the current framework, this research underwent several iterations to identify the most effective way to model the adversarial relationship between an attacker and a forensic auditor. Each stage revealed critical limitations that informed the final Stackelberg-based methodology.

4.1 Gridworld Analogy

The research began with a gridworld abstraction, where each tile represents a host or node in a network. In this environment, an attacker moves across tiles to steal a diamond (representing sensitive data) leaving a trace on every tile it visits. The guard moves from tile to tile seeking evidence of the attacker's existence. The attacker's goal is to exfiltrate the data and clean all traces while evading the guard.

4.2 FOND Iteration: The Scalability Wall

The second iteration utilized FOND planning. To simulate the uncertainty of an attacker's environment, the guard's movements were randomized, forcing the attacker to adopt risk-averse strategies. However, this approach suffers from severe state-space explosion. Gridworlds larger than 5x5 resulted in search times exceeding eight hours, making FOND impractical for anything beyond extremely simple tests.

4.3 Reinforcement Learning Iteration: The Guarantee Trade-off

To address scalability, I explored reinforcement learning. While RL agents can learn policy representations that scale better than exhaustive search methods, they introduce significant black box overhead, requiring extensive hyperparameter tuning for reward functions and success thresholds. Most importantly, RL lacks the formal guarantees of planning; whereas a planner

provides a mathematically verified answer for a given problem, an RL agent only provides a probabilistic approximation.

4.4 Stackelberg Iteration: The Balanced Approach

The final iteration reframed the gridworld as a Stackelberg planning task. By replacing non-deterministic obstacles with a strategic leader and a follower, the problem became computational tractable while remaining adversarially rigorous. This shift allowed for modeling the defender's proactive hardening of the environment against the attacker's optimal response.

5. Methodology

5.1 Formalizing the Anti-Forensics Attacker

The fundamental challenge in modeling a self-cleaning attacker lies in rigorously defining what constitutes "evidence" within a planning domain. In a standard cybersecurity environment, adversarial actions such as pivoting between machines via SSH or escalating privileges inevitably induce state mutations. These mutations manifest as modified system artifacts, such as authentication logs, altered file timestamps, or active process identifiers.

In static analysis, cleanliness is often environmentally specific. From the attacker's perspective, a trace left on a critical server may be fatal to an operation, whereas a trace on a peripheral node may be negligible. Manually encoding these acceptable end-states for every potential target environment is intractable and prone to human error. To address this, we introduce a formalism called the **restoration state**.

5.2 The Restoration State Mechanism

The restoration state is defined as the subset of the system's state space that must be reverted to its original configuration for an attack to be considered trace-free. Rather than manually selecting which logs to wipe, my methodology automates this by enforcing a constraint: the environment's state at the termination of the attack plan must be indistinguishable from the state at the initialization of the plan with respect to specific observable fluents.

Let F be the set of all possible system fluents and I be the initial state of the system. The restoration state, R, is defined as the subset of fluents that are true at initialization:

$$R = \{f \in F \mid I(f) = true\}$$

This set R encompasses the clean state of the environment.

5.3 Goal Augmentation Function

To enforce anti-forensic behavior within the planner, we must automatically augment the domain-specific goals. We transform the standard planning problem by appending the restoration state to the attacker's original objectives.

Let G represent the attacker's original goal set. The augmented goal set, G', is defined as the union of mission objectives and restoration constraints:

$$G' = G \cup R$$

This formulation forces the planner to find a valid plan π_A such that the final system state satisfies both the mission and the cleanup requirements. More granularly, we enforce a trace-clean-up constraint on the transition function δ^*:

$$R \subseteq \delta^*(I, \pi_A)$$

Where $\delta^*(I, \pi_A)$ represents the final state of the system after the execution of the attacker's plan π_A. This constraint ensures that for every action that negates a fluent in R (e.g., "dirtying" a log), the planner must schedule a corresponding remedial action to restore that fluent before the plan concludes.

5.4 Fluent Exclusion

A key nuance of the restoration state methodology is that not all initially true fluents are suitable for inclusion in the augmented goal set G'. This process involves identifying and filtering out specific fluents to ensure that the problem remains logically coherent. This exclusion isn't merely a performance optimization intended to reduce the state-space search; it is a

fundamental requirement for the restoration state methodology to function within a Stackelberg framework. If an agent modifies environmental factors that neither agent has a means of reverting, the planner will immediately deem the problem unsolvable, even if a valid attack path exists.

For example, in the network domain, the connectivity of hosts is defined by a *connected* fluent. If the defender disconnects two hosts as a hardening strategy, and neither agent possesses an action to re-establish that link, the *connected* fluent becomes an irreversible violation of the restoration goal. Similarly, including locational fluents like *at(agent, location)* in the restoration state would force the attacker to return to their entry point to satisfy the goal. While this models an exit constraint, it fundamentally alters the problem and may be incompatible with the desired goals.

There are two solutions to this problem, and both are equally sound. The first is requiring the programmer to invert the phrasing of the fluent. So instead of having a *connected* fluent, it would be renamed to *disconnected*. This way it would be initialized as false and excluded from the restoration state. Subsequently, all the action logic must be inverted to ensure logical consistency across the problem. This approach, however, becomes syntactically taxing on the programmer, as they are forced to mentally invert fluents when creating the initial problem. The second solution is much cleaner, which simply involves appending an identifier to the fluent name, like *static_*. Then, the augmentation function will ignore fluents containing the identifier. This preserves the initial logic of the domain and reduces the burden on the programmer.

6. Network Domain

To verify the correctness of my automated augmentation function and ensure its domain-independence, we transition from the abstract gridworld to a representative network domain modeled after standard enterprise infrastructure. This domain abstracts that infrastructure into three distinct segments, each with varying levels of trust and asset criticality.

6.1 Network Topology and Segmentation

The network is represented as an undirected graph where nodes represent hosts (personal computers) and edges represent valid SSH connections (Figure 1). The three logical segments are:

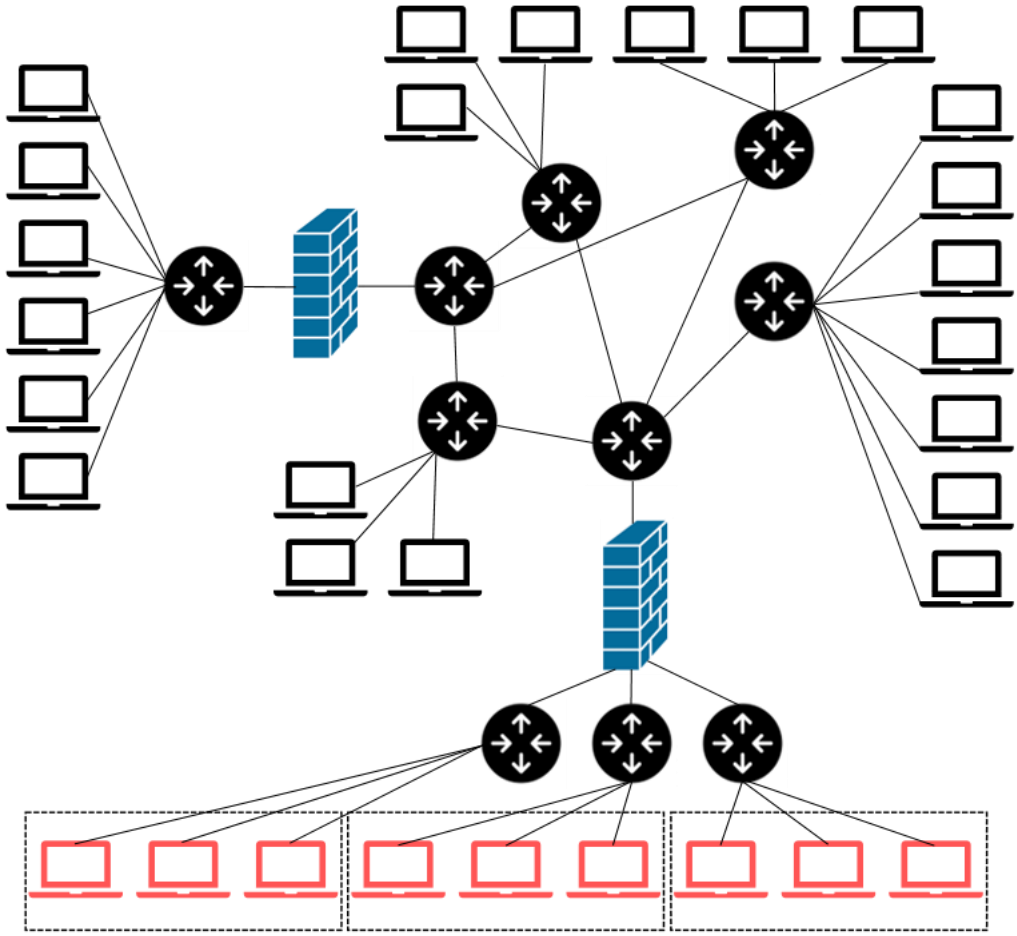

Figure 1: Logical network topology showing segmentation and connectivity

1. External Segment: The untrusted public-facing internet. This is the attacker's entry point.
2. Internal Segment: The organization's trusted internal network, containing employee workstations.
3. Critical Segment: The high-security zone hosting sensitive assets.

Connectivity between these segments is governed by two defense mechanisms. The external firewall mediates traffic between the external and internal segments. The internal firewall mediates traffic between the internal and critical segments. By default, the network topology allows a linear progression of access: External ↔ Internal ↔ Critical. Direct connections from the External segment to the Critical segment are strictly prohibited by movement preconditions.

6.2 Attacker Model

The attacker is modeled as a rational agent equipped with a predefined set of exploits and lateral movement capabilities defined by a PDDL action schema. The agent's lifecycle begins with an exploit action to gain user-level access on a target host, followed by privilege escalation to obtain root privileges. Once root access has been secured, the attacker can perform lateral movement to another host. Crucially, the domain enforces a strict constraint requiring root privileges on the source machine to initiate a connection to a destination machine. This movement explicitly models forensic artifacts; when the attacker transitions to a new host, the *logs_cleaned* fluent is negated, representing the generation of authentication logs and connection records. To counteract detection, the attacker utilizes the *clean* action, which restores the *logs_cleaned* fluent to true, effectively erasing evidence of their presence. The attacker's lifecycle culminates in reaching the Critical Segment, where they must execute the steal action to exfiltrate data before logging off to successfully escape the network.

6.3 Defender Model

In the Stackelberg planning framework, the defender acts as the leader, therefore the initial state of the problem counts as their first move. The defender has two main tools: patching specific vulnerabilities (CVEs) on individual machines and permanently closing SSH connections at the firewall. However, to keep the network operational, the defender is strictly forbidden to cut the final open connection between any two adjacent segments. By strategically combining these patches and firewall updates, the defender's goal is to completely block the attacker from reaching the critical segment, force them to take longer and more expensive detours, or trap them in routes where they do not have enough resources to clean their logs.

6.4 Success Conditions

The problem is defined as a zero-sum game between the attacker and the defender. Attacker success is achieved only if the agent can exfiltrate data from the critical segment and terminate the session while simultaneously satisfying all restoration constraints. Conversely, defender success is defined by the inability of the attacker to reach this state. The defender succeeds if the network configuration renders the attacker's goal unreachable due to infinite cost or if the defensive structure forces the attacker into a trajectory where satisfying the restoration constraint is impossible.

7. Results

The core of this research rests on two requirements: the system must augment different input domains correctly and the resulting plans must account for the attacker's need to erase their footprint. By testing the framework on both the abstract gridworld and the network domain, we verified that the methodology is both logically sound and domain-independent.

7.1 Defense Strategies

Before evaluating the defender's strategic interventions, it is critical to verify that the attacker's generated plans adhere to the restoration state constraints. The plan shown in Figure 2 illustrates a successful exfiltration mission within the gridworld domain prior to any defender intervention. Then, Figure 3 shows that the defender identifies critical bottlenecks and executes four *fix_place_wall* actions to physically obstruct the adversary's path. By placing these walls between specific tiles, the defender creates a state where the attacker's mission becomes *<unsolvable>*. The infinite attacker cost confirms that no sequence of movements allows for the exfiltration of the diamond while still satisfying the trace-cleaning requirements.

```
attacker plan:
        attack_move t00 t01 adversary_attacker
        attack_move t01 t02 adversary_attacker
        attack_move t02 t12 adversary_attacker
        attack_move t12 t11 adversary_attacker
        attack_move t11 t21 adversary_attacker
        attack_clean t21 adversary_attacker
        attack_steal t21 adversary_attacker
        attack_move t21 t11 adversary_attacker
        attack_clean t11 adversary_attacker
        attack_move t11 t12 adversary_attacker
        attack_clean t12 adversary_attacker
        attack_move t12 t02 adversary_attacker
        attack_clean t02 adversary_attacker
        attack_move t02 t01 adversary_attacker
        attack_clean t01 adversary_attacker
        attack_move t01 t00 adversary_attacker
        attack_clean t00 adversary_attacker
        attack_escape t00 adversary_attacker
```

Figure 2: An attack plan within the gridworld domain demonstrating a self-cleaning attacker.

```
fix ops costs: 4, attacker cost: 2147483647:
fix action sequences:
      sequence 0:
         fix_place_wall t10 t20 leader_defender
         fix_place_wall t10 t11 leader_defender
         fix_place_wall t01 t11 leader_defender
         fix_place_wall t01 t02 leader_defender
attacker plan:
      <unsolvable>
```

Figure 3: A defense plan within the gridworld domain demonstrating a sequence of actions which makes the attacker goals impossible.

Next, in the network domain, the attacker moves throughout the environment completing their goals while the restoration state is enforced (Figure 4). The defender utilizes a vulnerability-based hardening strategy rather than physical obstruction. The leader executes five fix_patch_cve actions targeting internal hosts pc6 through pc10 (Figure 5). This patching removes the specific entry points required for the attacker to progress through the network. The attacker goals become <unsolvable> indicating the defender closed all viable attack vectors. It should be noted that in a more realistic environment, there would be different costs associated with the defender actions. So simply patching all vulnerabilities may not be feasible all the time. However, those costs are extremely domain dependent, and for a generic viability test of this framework, equal costs will suffice.

Based on these successful plans, it is clear that the framework is truly domain-independent, as the same augmentation logic successfully handled both the gridworld and

network domain. Furthermore, the results verify that the attacker model is effectively anti-forensic, as the solver only accepts plans where the attacker can successfully reach the goal and execute the necessary remedial actions to restore the system state.

```
attacker plan:
       attack_move_ssh attacker_agent pc1 pc2
       attack_exploit attacker_agent pc2 ext_cve2
       attack_clean attacker_agent pc2
       attack_privilege_escalation attacker_agent pc2
       attack_move_ssh attacker_agent pc2 pc6
       attack_exploit attacker_agent pc6 int_cve1
       attack_clean attacker_agent pc6
       attack_privilege_escalation attacker_agent pc6
       attack_move_ssh attacker_agent pc6 pc11
       attack_steal attacker_agent pc11 critical_segment
       attack_clean attacker_agent pc11
       attack_log_off attacker_agent
```

Figure 2: An attack plan within the network domain demonstrating a self-cleaning attacker.

```
fix ops costs: 5, attacker cost: 2147483647:
fix action sequences:
      sequence 0:
         fix_patch_cve defender_agent pc9 int_cve4
         fix_patch_cve defender_agent pc8 int_cve3
         fix_patch_cve defender_agent pc7 int_cve2
         fix_patch_cve defender_agent pc6 int_cve1
         fix_patch_cve defender_agent pc10 int_cve1
attacker plan:
      <unsolvable>
```

Figure 5: A defense plan within the gridworld domain demonstrating a sequence of actions which makes the attacker goals impossible

7.2 Scalability Analysis

The scalability of this methodology is highly dependent on the structural complexity of the network domain. A network that is structurally simple but contains many nodes may scale significantly better than a small but highly complex architecture. The following analysis is intended to provide insight into the scaling behavior of this specific enterprise network model.

7.2.1 General Scaling in a Balanced Network

To establish a baseline, the framework's performance was evaluated on balanced network configurations where the number of nodes were distributed equally across the external, internal, and critical segments (Figure 6). As the number of nodes increases, the search time remains manageable until reaching the 7x7x7 configuration, which marks the first major exponential

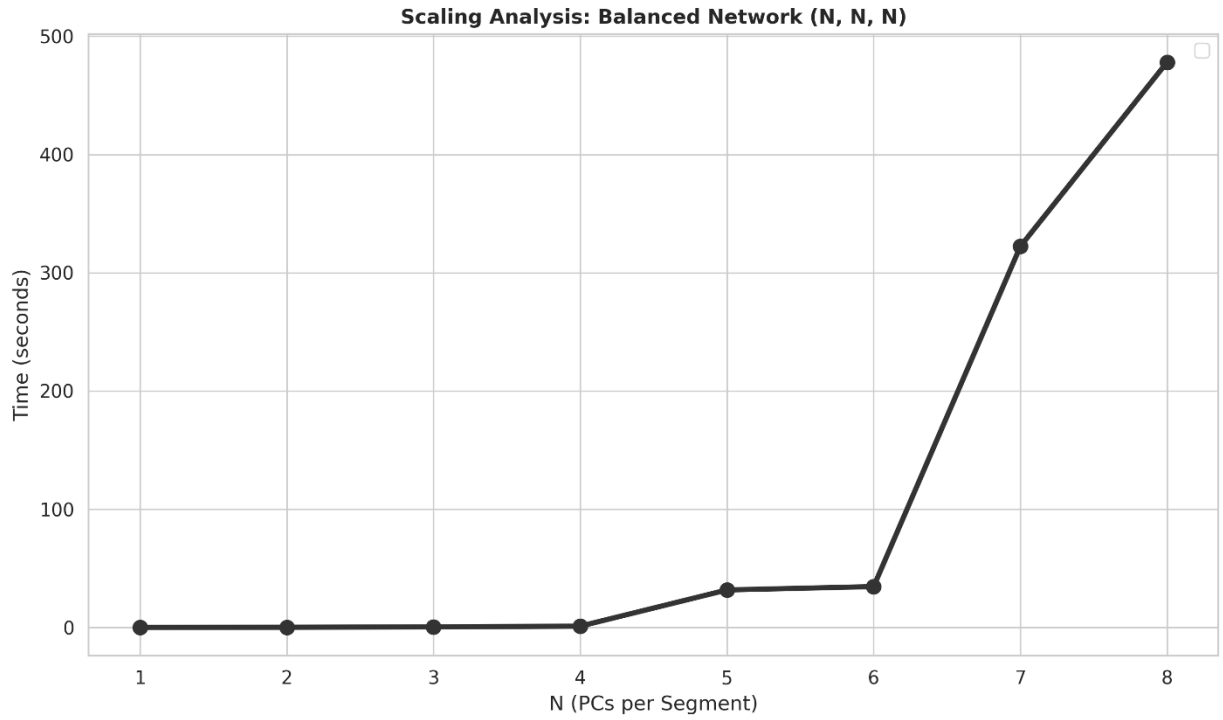

Figure 6: Illustrates the total search time as the number of nodes increases equally across all network segments. The final data point represents a failure to solve due to memory exhaustion.

jump, requiring approximately 322.3 seconds to solve. This is an 830.78% increase from the 6x6x6 configuration's 34.63 seconds to solve.

The subsequent 8x8x8 network configuration failed to find a solution. While the recorded failure time of approximately 478 seconds might appear unusually low for an exhaustive search, this was the result of the planner hitting the 4GB memory limit rather than exhausting the strict 30-minute compute timeout. This highlights that memory overhead, rather than pure compute time, can become the primary bottleneck when evaluating highly interconnected state spaces.

7.2.2 Segment Sensitivity

Beyond balanced scaling, an isolated sensitivity analysis reveals that certain network segments scale much better than others. In this experiment, we scale the number of hosts in one segment, while keeping the other two segments fixed at a single host (Figure 7). This revealed

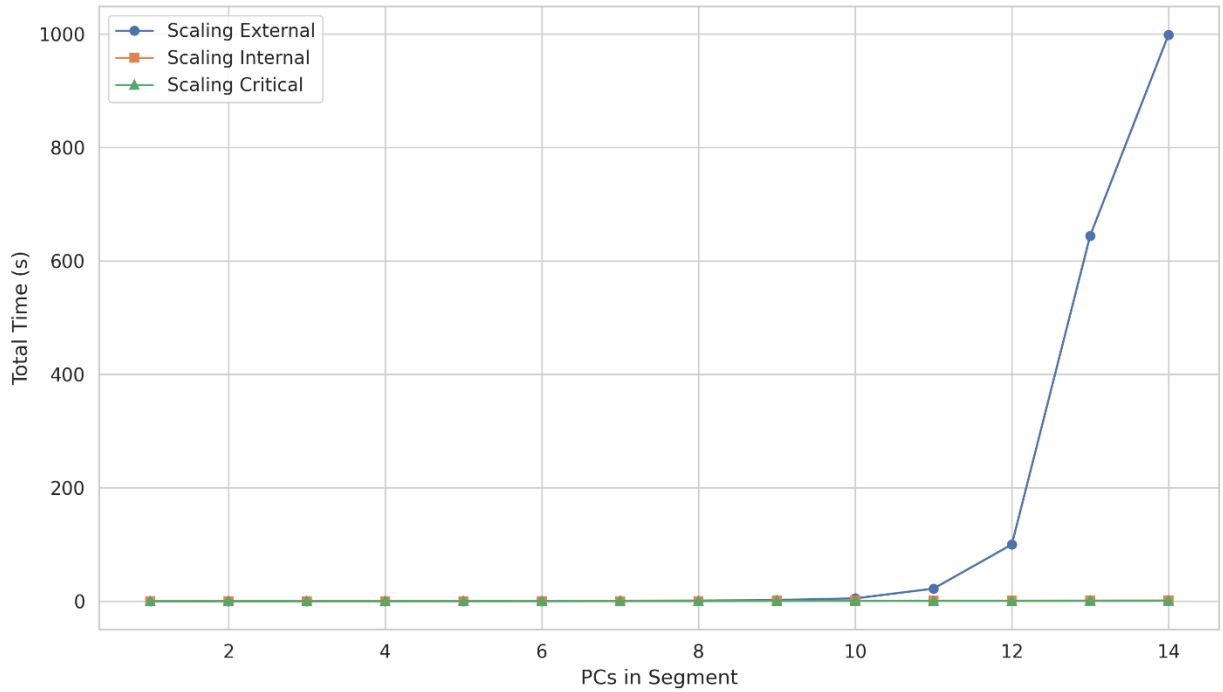

Figure 7: A multi-series plot comparing the impact of scaling a single network segment while keeping the others fixed at one host.

that increasing the size of the external segment degrades planner performance much faster than proportionally scaling the internal or critical segments.

This discrepancy is due to the external segment serving as the attacker's entry point. Adding nodes here creates a massive branching factor at the very top of the search tree, forcing the planner to evaluate exponentially more potential initial trajectories before reaching dead-ends or viable paths. By extrapolation, if a 4th network segment were introduced into the topology, the second shallowest segment would likely exhibit similar scaling issues as the branching complexity trickles down.

7.2.3 Combinatorial Complexity

Heatmap analysis of the various configurations visually confirms the exponential nature of the state-space explosion. The transition from computationally trivial configurations to highly demanding ones is marked by an abrupt threshold rather than a gradual linear increase. Furthermore, analyzing the most complex network configurations on the heatmap clearly illustrates the external segment's poor scaling relative to the internal segment.

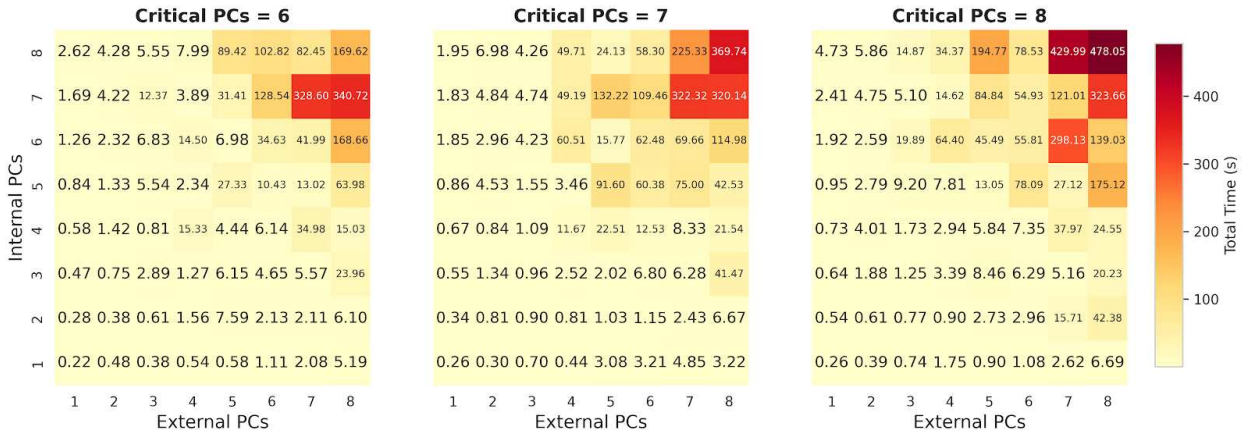

Figure 8: Three heatmaps visualizing the relationship between external and internal host counts across fixed critical segment sizes of 6, 7, and 8 hosts. The color gradient represents total search time.

8. Limitations & Future Work

The most immediate hurdle is scalability. The enterprise network modeled in this research consists of a relatively small number of hosts compared to production environments, which often encompass thousands of nodes. As demonstrated in the performance metrics, the state-space explosion remains a significant bottleneck, particularly as the number of initial entry points in the external segment increases. To alleviate this, future work should explore the integration of Lifted Stackelberg Planning. By utilizing symbolic reasoning over first-order logic schemas rather than grounded, instantiated states, lifted planners can often tackle problems orders of magnitude larger than traditional Stackelberg engines. Something to note is that comparing Stackelberg planning performance directly to other paradigms like FOND is a disanalogous comparison; while FOND handles uncertainty, it suffers from even more severe scaling issues in deterministic gridworlds, making Stackelberg the necessary baseline for simple scalability analysis in this domain.

Another limitation is that this framework currently requires some manual intervention from the programmer to function correctly. Specifically, making syntactic fluent adjustments for the restoration state to be correctly instantiated, and when incorporating cleaning actions. Future iterations could benefit from dynamic model acquisition. Integrating tools that automatically generate PDDL representations from live network scans would reduce human error and allow the framework to adapt to changing network topologies with less manual intervention.

Finally, the efficacy of defense is strictly bottlenecked by the fidelity of the programmer defined domain. In an automated planning environment, the engine acts as a perfect adversary within the provided rules; if there are any logical gaps or overlooked causal chains in the PDDL, the planner will exploit them. This creates a bit of a paradoxical challenge. High fidelity leads to

more realistic security verification but worsens the scalability due to a larger search space. Low fidelity scales well but may leave the network vulnerable to unaccounted for attack vectors.

Performing a comprehensive cross-domain analysis remains difficult because any given cybersecurity problem can be expressed in an infinite number of ways. Even with grounding, the way an architect chooses to represent host connectivity or privilege levels fundamentally changes the search tree. Standardizing these representations is a necessary step for benchmarking anti-forensic modeling across different industrial sectors.

9. Conclusion

This research demonstrates that integrating Automated Planning with a Stackelberg game-theoretic framework provides a rigorous method for characterizing the behavior of an anti-forensic attacker. By formalizing the restoration state mechanism, we have enabled the automated generation of attack plans where success is strictly contingent upon the removal of detectable footprints. The methodology proved to be domain-independent, successfully verifying security invariants across both abstract gridworld environments and representative network topologies.

The results highlight a fundamental shift in defensive strategy: rather than pursuing a potentially unattainable perfect defense, network architects can use this framework to design environments that force adversaries into an inescapable "trace-or-fail" dilemma. While scalability remains a challenge as network complexity grows, the use of Stackelberg search provides a computationally tractable path forward for proactive network hardening. Ultimately, this framework serves as a foundation for building more resilient digital infrastructures that are hardened not only against initial intrusion but against the sophisticated erasure techniques of modern autonomous threats.

Bibliography

[1] Achuthan, K., Ramanathan, S., Srinivas, S., & Raman, R. (2024). "Advancing cybersecurity and privacy with artificial intelligence: current trends and future research directions." *Frontiers in Big Data*, 7.

[2] Ali, S. M., Razzaque, A., Yousaf, M., & Shan, R. U. (2025). "An Automated Compliance Framework for Critical Infrastructure Security Through Artificial Intelligence." *IEEE Access*, 13, 4436-4459.

[3] Boddy, M., et al. (2005). "Course of Action Generation for Cyber Security Using Classical Planning." *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

[4] Cates, B., Kulkarni, A., and Sreedharan, S. (2023). "Planning for Attacker Entrapment in Adversarial Settings." *arXiv preprint arXiv:2303.00822*.

[5] Ghanem, M. C., Chen, T. M., and Fernandez, E. B. (2024). "The Future of Autonomous Forensic Agents and AI-Augmented Investigators." *ResearchGate*.

[6] Hoffmann, J. (2015). "Simulated Penetration Testing: From 'Dijkstra' to 'Turing Test++'." *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

[7] Karnouskos, S. (2011). "Stuxnet worm impact on industrial cyber-physical system security." *IECON 2011-37th Annual Conference of the IEEE Industrial Electronics Society*, 4490-4494.

[8] Lucangeli, J., et al. (2013). "Attack Planning in the Real World." *arXiv preprint arXiv:1306.4044*.

[9] McDermott, D., et al. (1998). "PDDL - The Planning Domain Definition Language." *Technical Report CVC TR-98-003/DCS TR-1165*, Yale Center for Algorithms and Complexity.

[10] Ren, Y., et al. (2024). "Automated tactics planning for cyber attack and defense based on large language model agents." *PubMed*.

[11] Ryan, M. (2021). "Ransomware Revolution: The Rise of a Prodigious Cyber Threat." *Advances in Information Security*.

[12] Sauer, N., et al. (2023). "Lifted Stackelberg Planning." *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

[13] Schneier, B. (1999). "Attack Trees." Dr. Dobb's Journal.

[14] Sheyner, O., et al. (2002). "Automated generation and analysis of attack graphs." *IEEE Symposium on Security and Privacy*, 273-284.

[15] Speicher, P., et al. (2018). "Stackelberg Planning: Towards Effective Leader-Follower State Space Search." *Proceedings of the AAAI Conference on Artificial Intelligence*.

[16] Standen, M., et al. (2021). "CybORG: A Gym for the Development of Autonomous Cyber Agents." *arXiv preprint arXiv:2108.09118*.

[17] Tarwireyi, P., Terzoli, A., & Adigun, M. O. (2024). "Meta-SonifiedDroid: Metaheuristics for Optimizing Sonified Android Malware Detection." *IEEE Access*, 12, 134779-134808.

[18] Torralba, A., et al. (2021). "Faster Stackelberg Planning via Symbolic Search and Information Sharing." *Proceedings of the AAAI Conference on Artificial Intelligence*.

[19] Zhao, Y., Ge, Y., & Zhu, Q. (2021). "Combating Ransomware in Internet of Things: A Games-in-Games Approach for Cross-Layer Cyber Defense and Security Investment." *Lecture Notes in Computer Science*, 208-228.