# SPEAR: Security Posture Evaluation using AI Planner-Reasoning on Attack-Connectivity Hypergraphs


Rakesh Podder
Colorado State University
Fort Collins, Colorado, USA
rakesh.podder@colostate.edu

Turgay Caglar
Colorado State University
Fort Collins, Colorado, USA
turgay.caglar@colostate.edu

Shadaab Kawnain Bashir
Colorado State University
Fort Collins, Colorado, USA
shadaab.bashir@colostate.edu

Sarath Sreedharan
Colorado State University
Fort Collins, Colorado, USA
sarath.sreedharan@colostate.edu

Indrajit Ray
Colorado State University
Fort Collins, Colorado, USA
indrajit.ray@colostate.edu

Indrakshi Ray
Colorado State University
Fort Collins, Colorado, USA
indrakshi.ray@colostate.edu



## Abstract

Graph-based frameworks are often used in network hardening to help a cyber defender understand how a network can be attacked and how the best defenses can be deployed. However, incorporating network connectivity parameters in the attack graph, reasoning about the attack graph when we do not have access to complete information, providing system administrator suggestions in an understandable format, and allowing them to do what-if analysis on various scenarios and attacker motives is still missing. We fill this gap by presenting SPEAR, a formal framework with tool support for security posture evaluation and analysis that keeps human-in-the-loop. SPEAR uses the causal formalism of AI planning to model vulnerabilities and configurations in a networked system. It automatically converts network configurations and vulnerability descriptions into planning models expressed in the Planning Domain Definition Language (PDDL). SPEAR identifies a set of diverse security hardening strategies that can be presented in a manner understandable to the domain expert. These allow the administrator to explore the network hardening solution space in a systematic fashion and help evaluate the impact and compare the different solutions.



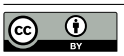
## CCS Concepts

• **Security and privacy** → *Distributed systems security*; **Network security**; • **Networks** → **Network security**.

## Keywords

Attack-Connectivity Graph, AI Planning, Network Hardening, Attack Graph Analysis

**ACM Reference Format:**
Rakesh Podder, Turgay Caglar, Shadaab Kawnain Bashir, Sarath Sreedharan, Indrajit Ray, and Indrakshi Ray. 2025. SPEAR: Security Posture Evaluation using AI Planner-Reasoning on Attack-Connectivity Hypergraphs. In *Proceedings of the 30th ACM Symposium on Access Control Models and Technologies (SACMAT '25), July 8–10, 2025, Stony Brook, NY, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3734436.3734451



*SACMAT '25, Stony Brook, NY, USA*



ACM ISBN 979-8-4007-1503-7/2025/07
https://doi.org/10.1145/3734436.3734451


## 1 Introduction

Cyber attacks continue to pose a significant threat despite increased efforts focusing on network security. One approach for understanding and mitigating network vulnerabilities is attack graph analysis. An attack graph (also known as a cybersecurity graph, threat graph, exploitability graph, vulnerability graph, or risk-assessment graph) represents the vulnerabilities present in a system and dependencies among those vulnerabilities. The graph helps identify the order and the combination in which vulnerabilities must be exploited to launch the attack. It provides security professionals with valuable insights into potential attack vectors and can help prioritize efforts to secure critical assets effectively.

Schneier [33] was the first to propose the paradigm of attack trees for modeling security threats. Since then attack trees/graphs have been variously discussed for different aspects of cyber preparation including cyber risk management, determining least cost paths for attacks, and cost-benefit analysis of cyber hardening measures [3, 9–11, 23, 27, 28, 30, 33, 38]. However, there are several gaps related to the use of attack graphs. Our proposed graph, known as *Attack-Connectivity Graph* (*ACG*), generalizes the earlier works and alleviates their shortcomings. First, we are not aware of any attack graph framework that allows the users to perform **what-if analysis on attack graphs driven by feedback about potential defensive strategies and their efficacy**. Second, almost all previous works assume that the attack graph is monotonic [15]. The monotonicity assumption translates to the condition that the attacker is exploiting an increasing number of vulnerabilities and thus increasing its privileges as it penetrates further into the network. Since there are only a polynomial number of privileges an attacker can gain, *this assumption enables analysis algorithms to terminate in polynomial time.* However, this may not always be true. For example, the monotonicity assumption precludes an attacker from revisiting nodes previously visited – a distinct possibility for lateral movements. If the attacker can only launch attacks from specific hosts, this scenario cannot be modeled with the monotonicity assumption. **We relax the monotonicity assumption but still provide a scalable solution**. Third, most attack graph tools' are focused on identifying attack paths and defensive strategies. However, these do not evaluate the effect of these strategies on network properties such as, but not limited to, connectivity. **By jointly modeling both dependencies among vulnerabilities and other network properties in the form of a hypergraph,**

**we enable such analysis.** Last but not least, most existing tools fail to support model reusability. As a system changes, the corresponding attack graph also changes. Incremental updates and reuse of attack graphs are poorly supported by most existing tools. This is a unique feature of work approach.

We present SPEAR , a formal framework to evaluate security posture using AI Planning for attack graph modeling. It uses the *causal formalism* of AI planning to model vulnerabilities and configurations in a networked system. This causal representation is very close to *how human users reason about actions* [20] and enable the user to guide the SPEAR tool for performing refined what-if analysis. Therein lies the advantage of SPEAR over logic-based frameworks such as MulVAL [27] which requires a different type of expertise than can be expected from system administrators as well as other AI planning based cybersecurity frameworks [5, 14, 36]. SPEAR uses AI planning techniques to identify potential attack paths (or plans in AI planning parlance) in the network. As AI planning is scalable to very large problems, SPEAR can be used for real-world networks. Once attack paths are identified, SPEAR automatically provides defensive strategies for cyber-hardening the network. Cyber-hardening in SPEAR (or preventing the attacker from reaching the goal node in the attack graph) is reduced to the problem of rendering an attack graph "unsolvable" (in AI Planning parlance unsolvable means no attack path is found) or increasing the minimum cost of a plan for potential attackers. Hardening the network to prevent the attacker from reaching the goal node in the attack graph could involve changes to the network that might be quite expensive. Additionally, in most practical situation, it might be hard to estimate the cost associated with these changes upfront. As such, instead of simply finding a single set of changes, we identify *a set of diverse solutions* that achieves the desired level of security readiness but involves updating different parts of the overall network.

These solutions involve strategically modifying the network configuration, introducing new defense mechanisms, or strengthening existing ones. By considering the model space search, we aim to identify the *most effective combination of changes* that disrupt attack paths; the system-admin can choose the solution they believe would be the easiest to implement for their specific use case. This approach is different from attack graph analysis tools that model the network hardening problem as a multi-objective optimization problem [9, 12, 22, 29] in that this technique allows the defender to perform what-if analysis and select the most desirable solution. In this work, we make the following major contributions:
(i) Introduce the notion of Attack-Connectivity Graph (*ACG*) that allows us to jointly model both attack paths and network connectivity. Relaxed the monotonicity assumptions. Showed how *ACG* can be captured using AI planning and provide complexity results.
(ii) Illustrate how this model can be used to analyze the robustness of the underlying network using various graph-theoretic metrics. We focus on *impenetrability* and *attack difficulty* metrics and show how these metrics can be calculated for a given *ACG* using modified planning representations of the original graphs.
(iii) Propose a design framework that allows us to perform what-if analyses to identify network updates that could harden the network as defined by the different metrics.
(iv) Show how we calculate such designs using a heuristic search over the space of model edits, and we also propose a novel admissible heuristic that can still guarantee identifying optimal solutions.
(v) Demonstrated how we can use this framework when the design costs are partially unspecified by generating diverse solutions.
(vi) Perform a series of empirical evaluations on networks of different configurations and scales to test the computational characteristics of our proposed algorithm.

The rest of this paper is organized as follows. Section 2 provides a brief introduction to AI planning. Section 3 introduces the Attack-Connectivity Graph (*ACG*) model. Section 4 describes how SPEAR tool can leverage *ACG* for analyzing network security. Section 5 discusses the architecture of SPEAR and the test network. Section 6 presents a running example, focusing on the ability of the SPEAR tool to perform what-if analysis on the test network. Section 7 presents an empirical evaluation of SPEAR tool's performance. Section 8 discusses related work. Section 9 concludes the paper.

## 2 Background

A classical planning problem is defined by a tuple $\mathcal{M} = \langle F, A, I, G \rangle$, where $F$ corresponds to a set of state variables or fluents, $A$ the set of actions, the initial state $I$, and goal specification $G \subseteq F$. A given planning problem is a compact representation of a directed graph called a transition system. The nodes in this graph correspond to the different states of the system. Each state $s$ is characterized by the values of a set of propositional (boolean) variables denoted by $F$. The directed edges in the graph correspond to transitions between states as a result of executing an action. Each action, $a \in A$, is further defined by $a = \langle pre(a), add(a), del(a) \rangle$; where $pre(a) \subseteq F$ is the precondition that determines the states where an action can be executed, $add(a) \subseteq F$ are the add effects that determine the fluents that will be set true by the execution of the action and $del(a) \subseteq F$ are the delete effects that determine the fluents that will be set false by the execution of the action. The result of executing an action at a state $s$, will be captured by a transition function $\Gamma_{\mathcal{M}}$, such that $\Gamma_{\mathcal{M}}(s, a) = (s \setminus del(a)) \cup add(a),\ if\ pre(a) \subseteq s$. We overload the notation and also use $\Gamma_{\mathcal{M}}$ to capture execution of action sequence, such that $\Gamma_{\mathcal{M}}(s, \langle a_1, ..., a_k \rangle) = \Gamma_{\mathcal{M}}(\Gamma_{\mathcal{M}}(...\Gamma_{\mathcal{M}}(s, a_1)..., a_{k-1}), a_k)$. The problem of planning is to find a path from the initial state to a goal state. Specifically, a solution to a planning problem is called a plan, and it takes the form of a sequence of actions. An action sequence $\pi = \langle a_1, ..., a_k \rangle$ is said to be a valid plan if $\Gamma_{\mathcal{M}}(I, \pi) \supseteq G$. Assuming each action has a unit cost, a plan is optimal if no shorter valid plans exist.

## 3 Attack-Connectivity Graph Formalization

We assume a network consisting of a set of hosts/machines (uniquely identified by a label) laid out in a particular network topology. In *ACG*, we represent a network using a set of propositional facts referred to as attributes (that take in boolean values)[1]. The attributes capture host-level information, including information about host labels, software, versions, configuration, functions, vulnerabilities, whether the attacker has access to it, etc. The network-level information, specifically, direct connection between nodes is denoted by a connectivity function $\phi_C$.

[1] Multi-valued variables can easily be turned into boolean ones

Definition 1. *The attribute set $P_{\mathcal{H}}$ is a set of propositions that could be true for some host in a given network. We denote the set of unique hosts and their labels using $\mathcal{H}$. The connectivity between hosts in a given network is captured by the connectivity function $\phi_C : \mathcal{H} \times \mathcal{H} \rightarrow \{True, False\}$, where the function returns true for a pair of hosts if they are directly connected in the network.*

We represent an attack using pre-conditions and post-conditions. Pre-conditions are conditions needed for the attack to succeed. Post-conditions are conditions satisfied after the execution of the attack. An attack's pre and post-conditions are represented using subsets of host-level attributes. Note that each attack may be associated with multiple pre-conditions and might elicit different post-conditions based on the pre-conditions. This is one of the reasons why attack graphs are popularly modeled as AND-OR graphs [9]. We formalize attacks as follows.

Definition 2. *Let $\mathcal{A}$ be the set of attacks associated with a network, where each attack $a \in \mathcal{A}$, is captured by a set of the form $a = \{\langle pre^1(a), post^{+1}(a), post^{-1}(a)\rangle, ..., \langle pre^k(a), post^{+k}(a), post^{-k}(a)\rangle\}$, where for any attack tuple, $pre^i(a) \subseteq P_{\mathcal{H}}$ captures the preconditions, $post^{+i}(a) \subseteq P_{\mathcal{H}}$ captures the positive postconditions (i.e., the set of attributes turned true), and $post^{-i}(a) \subseteq P_{\mathcal{H}}$ captures the negative postconditions (i.e., the set of attributes turned false).*

Our proposed hypergraph generalizes AND-OR graphs. Pre-conditions are set by multiple nodes, whereas post-conditions are set by only the attack node. The nodes in this graph correspond to all the possible configurations a host can take, and the edges correspond to possible network connections.

Definition 3. *An Attack-Connectivity Graph $\mathcal{G}$ denoted as $\mathcal{G} = \langle \mathcal{N}, \mathcal{E}\rangle$, where $\mathcal{N}$ are the nodes and $\mathcal{E}$ are the edges. Nodes represent hosts along with their attributes, namely, $\mathcal{N} = \mathcal{H} \times 2^{P_{\mathcal{H}}}$, and edges are either connectivity edges or attack hyperedges, that is, $\mathcal{E} = \mathcal{E}_C \cup \mathcal{E}_{\mathcal{A}}$ where $\mathcal{E}_C$ are the connectivity edges, and $\mathcal{E}_{\mathcal{A}}$ are the attack hyperedges.*
*Connectivity edges $\mathcal{E}_C$: A connectivity edge $e \in \mathcal{E}_C$ exists between nodes $n_1, n_2 \in \mathcal{N}$ where $n_1 = (h_1, P_1)$ and $n_2 = (h_2, P_2)$, if $\phi_C(h_1, h_2) = True$. $src(e) = \{n_1\}$ and $dest(e) = \{n_2\}$ denote the source and destination of edge e respectively.*
*Attack hyperedges $\mathcal{E}_{\mathcal{A}}$: An attack hyperedge $e \in \mathcal{E}_{\mathcal{A}}$ of the form $src(e) = \{n_1, ..., n_k\}$ and $dest(e) = \{n'_i\}$ exists iff there exists attack $a_i = \langle pre^i(a), post^{+i}(a), post^{-i}(a)\rangle$, where*

- *$\exists n_i \in src(e)$, that corresponds to the same host as the one for $n'_i \in dest(e)$, where $n_i = (h_i, P_i)$*
- *$pre^i(a) \subseteq \bigcup_{n_j \in src(e)} P_j$, where $n_j = (h_j, P_j)$*
- *for the node $n'_i = (h_i, P'_i)$, we have $P'_i = (P_i \setminus post^{-i}(a)) \cup post^{+i}(a)$*
- *$\forall h_j \in src(e)$, $h_j = h_i$ or $\phi_C(h_i, h_j) = True$*

Note, $\mathcal{G}$ is exponential in the size of the set of attributes ($P_{\mathcal{H}}$). The nodes associate every possible proper subset of attributes to each host, capturing the changes that may be brought about in the node due to various attacks. Our graphs can also have cycles. We use this joint modeling to analyze the security posture and connectivity of the network simultaneously and do a what-if analysis of the network configuration. Note that, our formulation allows us to replace connectivity with other, more sophisticated notions of availability, including the availability of specific services.

We now introduce the notion of transition function $\lambda$ over the *ACG*. Given a set of nodes $S_N \subseteq \mathcal{N}$, and an edge $e \in \mathcal{E}$, the transition function describes the result of following that edge. Specifically, we define it as:

$$\lambda(S_N, e) = \begin{cases} S_N \cup dest(e) \text{ if } src(e) \subseteq S_N \text{ and } e \in \mathcal{E}_C \\ S_N \setminus (h_i, P_i) \cup dest(e) \text{ if } src(e) \subseteq S_N,\ e \in \mathcal{E}_{\mathcal{A}} \\ \textit{undefined} \text{ otherwise} \end{cases}$$

where $(h_i, P_i)$ is the original state of the host affected by the attack. We can also extend this definition to apply to a sequence of edges. We now define valid paths below.

Definition 4. *For a given* ACG $\mathcal{G}$*, and set of initial nodes $S^0_N$, an edge-sequence $\mathbf{E} = \langle e_1, ....., e_n\rangle$ is considered:*

- *A* valid attack path *to a target node attribute $(h_i, p_i)$ if and only if $(h_i, p_i) \in \bigcup_{(h,P)\in\lambda(S^0_N,\mathbf{E})}\{(h, p)|p \in P\}$,*
- *A* valid connectivity path *to a target node $h_i$ if and only if $n \in \lambda(S^0_N, \mathbf{E})$, such that $n = (h, P)$.*

All analysis performed on our *ACG* will be represented in terms of the presence or absence of these two different paths. However, instead of performing the analysis directly on graph $\mathcal{G}$, we will first compile them into a planning problem and use fast planners to perform the analysis.

Definition 5. *For a given* ACG $\mathcal{G}$*, initial node set $S^0_N$, and target attribute $(h_t, p_t)$, a representative planning model is defined as $\mathcal{M}^{\mathcal{G}} = \langle F^{\mathcal{G}}, A^{\mathcal{G}}, I^{\mathcal{G}}, G^{\mathcal{G}}\rangle$, where*

- *$F^{\mathcal{G}}$ contains a fluent for each host attribute pair in $\mathcal{H} \times P_{\mathcal{H}}$, where we use function $\gamma^{\mathcal{M}}$ to capture the mapping from $\mathcal{H} \times P_{\mathcal{H}}$ to $F^{\mathcal{G}}$ (to simplify notation we will also allow the application of the function over sets).*
- *$A^{\mathcal{G}}$ contains an action for each edge in $e \in \mathcal{E}$. Let $a_e$ be the action corresponding to an edge e with destination node $(h_i, P_i)$. Here the action precondition is given as $pre(a_e) = \bigcup_{(h,P)\in src(e)} \gamma^{\mathcal{M}}(\{(h, p) \mid p \in P\})$. If it is an attack edge, corresponding to an attack tuple, $\langle pre, post^+, post^-\rangle$, then $add(a_e) = \gamma^{\mathcal{M}}(\{h_i\} \times post^+)$, and $del(a_e) = \gamma^{\mathcal{M}}(\{h_i\} \times post^-)$. In the case of a connectivity edge, the add effect is given as $add(a_e) = \gamma^{\mathcal{M}}(\{h_i\} \times P_i)$, and the delete is empty, i.e., $del(a_e) = \emptyset$.*
- *$I^{\mathcal{G}} = \bigcup_{(h,P)\in S^0_N} \gamma^{\mathcal{M}}(\{h, p \mid p \in P\})$*
- *$G^{\mathcal{G}} = \{\gamma^{\mathcal{M}}((h_t, p_t)\}$.*

Now, we need to show that our model representation is in fact a lossless representation of the true *ACG*.

Theorem 1. *For a given* ACG $\mathcal{G}$*, initial node set $S^0_N$, and target node $n_t$, the representative planning model $\mathcal{M}^{\mathcal{G}}$, is a sound and complete representations:*

*C1 It is sound so far that any valid plan in $\mathcal{M}^{\mathcal{G}}$ must correspond to a valid connectivity or attack path in $\mathcal{G}$.*
*C2 It is complete so far that any valid connectivity or attack path in $\mathcal{G}$ must correspond to a valid plan in $\mathcal{M}^{\mathcal{G}}$.*

Proof Sketch. The proof follows from the observation that each action directly corresponds to an edge. An action is only

executable in a state if the fluents corresponding to the source nodes of the equivalent edge are present in the state. The result of executing an action is making the fluent corresponding to the destination nodes true. Similarly, we can prove completeness by showing that each path through the $\mathcal{G}$ can be mapped over to the plans in $\mathcal{M}^{\mathcal{G}}$. □

This equivalence also allows us to prove the complexity of finding a path through *ACG*:

Theorem 2. *Given the attribute set $P_{\mathcal{H}}$, the set of host $\mathcal{H}$, connectivity function $\phi_C$, and attack set $\mathcal{A}$,*

*(1) The problem of valid attack path existence in the corresponding* ACG *is PSPACE-complete.*
*(2) The problem of valid connectivity path existence in the corresponding* ACG *is polynomial.*

Proof Sketch. In the case of attack paths, the membership proof, i.e., the problem is within the class of PSPACE, is given by Theorem 1, and the fact that the plan existence problem is PSPACE-complete for positive pre-condition STRIPS planning [7]. The model described above belongs to that class, proving the membership.

Regarding hardness, we need to show that for any planning problem, we can polynomially reduce it to a problem of identifying an attack path. Particularly, we identify an attribute set $P_{\mathcal{H}}$, a set of hosts $\mathcal{H}$, a connectivity function $\phi_C$, and attack set $\mathcal{A}$, such that an attack path exists in the corresponding *ACG* $\mathcal{G}$, only if there is a plan in the corresponding model. Here, we can set the attributes to be equivalent to the fluents in the original model, have a single host (thus, the connectivity function is irrelevant), and the attack definition is equivalent to the action definitions. This means the transition function $\lambda$ is exactly equal to the transition function of the corresponding planning problem. As such, there is only an attack path if there is a corresponding plan. In the most general case, the planning problem could have a multi-fluent goal description. We can translate that into the single host attribute by introducing a new attack, whose pre-conditions are the attributes corresponding to the original goal, and the post-condition is a new attribute used as the target.

Finally, for the connectivity path, the corresponding actions have an empty delete effect. Note, the existence of connectivity paths can be performed by only considering delete-free actions. It has been shown that the problem of identifying plan existence in the presence of delete-free actions is polynomial in complexity [16]. □

# 4 SPEAR : Security Posture Evaluation using AI Planner-Reasoning

We now look at how *ACG* can be used as a basis to evaluate security posture for network analysis. The framework is instructable in that, given a starting network, the administrator, can instruct the system to identify ways to update the network so that it meets certain specified requirements. Once the system finds a set of changes, it will present these solutions to the administrator that meet the specified requirements. Below, we discuss how to specify these requirements, how to find a set of network updates that will ensure the satisfaction of specified requirements, and how one can explain why these updates help achieve the requirements.

## 4.1 Metrics for Analysis

In this case, we assume that the user requirements are specified with respect to some metrics related to potential attack paths and connectivity paths. The user may have a goal of having the metrics take some specific value. Before we get into the details, we provide a general definition of *ACG* metric.

Definition 6. *An analytic metric for* ACG *is given as a pair of functions $\langle \mathcal{F}_{\mathcal{A}}, \mathcal{F}_C \rangle$. For a given graph $\mathcal{G}$, a set of initial nodes $\mathbb{S}_N^0 = \{S_1^0, ..., S_k^0\}$, a set of attack targets $\mathbb{S}_t^{\mathcal{A}}$ and a set of connectivity targets $\mathbb{S}_t^C$, $\mathcal{F}_{\mathcal{A}}$ takes the space of all valid attack paths between initial sets from $\mathbb{S}_N^0$ to targets in $\mathbb{S}_t^{\mathcal{A}}$ (i.e., $\mathbb{E}_{\mathcal{A}}$) as input and returns a real number ($\mathcal{F}_{\mathcal{A}} : \mathbb{E}_{\mathcal{A}} \mapsto j, j \in \mathbb{R}$). Similarly, $\mathcal{F}_C$ takes as input the set of all connectivity paths ($\mathbb{E}_C$) to connectivity targets and returns a real number.*

Our use of initial node sets and target sets allows the analytic metric to be applied to analyze a set of attack and service conditions. As defined currently, the metrics may include any functions. They become more meaningful when we consider desirable properties for the network.

Definition 7. *An analytic metric pair $\langle \mathcal{F}_{\mathcal{A}}, \mathcal{F}_C \rangle$, is called an robustness metric if $\mathcal{F}_{\mathcal{A}}$ is a monotonically increasing function with respect to decreasing attack paths and $\mathcal{F}_C$ is monotonically increasing function with respect to increasing connectivity paths, Or*

- $\mathcal{F}_{\mathcal{A}}(\mathbb{E}_1) \geq \mathcal{F}_{\mathcal{A}}(\mathbb{E}_2)$, *if and only if,* $\mathbb{E}_1 \subseteq \mathbb{E}_2$.
- $\mathcal{F}_C(\mathbb{E}_1) \geq \mathcal{F}_C(\mathbb{E}_2)$, *if and only if,* $\mathbb{E}_1 \supseteq \mathbb{E}_2$.

We see that $\mathcal{F}_{\mathcal{A}}$ increases as attack paths are removed and $\mathcal{F}_C$ increases with connectivity. This formulation naturally lends itself to a multi-objective framework. However, we consider simpler settings where $\mathcal{F}_C$ is effectively an indicator function. In particular, we consider two specific robustness metrics. The first captures cases where we do not want to allow any attacks on the target, and the second tries to increase the attack hardness (measured in terms of the number of attacks to be carried out using the shortest attack path). In each case, we make sure that there exists at least one connectivity path. For notational convenience, we define the connectivity function as $\mathcal{F}_C^1$:

$$\mathcal{F}_C^1(\mathbb{E}_C) = \begin{cases} 1, & \text{if there exists a path to each connectivity target} \\ 0, & \text{otherwise} \end{cases}$$

Definition 8. *An impenetrability metric is given by the pair $\langle \mathcal{F}_{\mathcal{A}}^{\mathcal{I}}, \mathcal{F}_C^1 \rangle$, such that*

$$\mathcal{F}_{\mathcal{A}}^{\mathcal{I}}(\mathbb{E}_{\mathcal{A}}) = \begin{cases} 1, & \textit{if } |\mathbb{E}_{\mathcal{A}}| = 0 \\ 0, & \textit{otherwise.} \end{cases}$$

Definition 9. *An attack difficulty metric is given by the pair $\langle \mathcal{F}_{\mathcal{A}}^{\mathcal{D}}, \mathcal{F}_C^1 \rangle$, such that*

$$\mathcal{F}_{\mathcal{A}}^{\mathcal{D}}(\mathbb{E}_{\mathcal{A}}) = \begin{cases} \min\{|\mathbf{E}| \mid \mathbf{E} \in \mathbb{E}_{\mathcal{A}}\} & \textit{if } |\mathbb{E}_{\mathcal{A}}| > 0 \\ |\mathcal{N}| + 1, & \textit{otherwise.} \end{cases}$$

*where $|\mathcal{N}|$ is an upper bound on the longest possible attack path length.*

Now, we will see how we can take a given network and identify changes that would allow the network to meet the requirements defined with respect to these metrics.

## 4.2 Identifying Diverse Model Updates

In this section, our focus is to identify ways in which we can modify a given graph to ensure that the specified metric takes a given value. For the two metrics that were defined, the primary way to improve them is to remove potential attack paths while making sure that changes leave, at the very least, a single connectivity path. To keep the discussion relatively simple, we focus on cases where we have a single set of initial state nodes, a single attack target, and a connectivity target. However, the methods discussed are easily extensible to multiple sets of initial and target sets. None of the identified theoretical properties change for the more general case.

As mentioned earlier, we will perform such analysis using the equivalent representative planning model. To perform such analyses, we need to formalize the notion of model update. We start this formalization by first defining a model parameterization function that converts a given model to a set of features. In particular, we follow the conventions set in earlier model reconciliation papers [34, 35] and define a function $\delta$ as follows.

Definition 10. *The model parameterization function $\delta$ maps a given model $\mathcal{M} = \langle F, A, I, G\rangle$ to a subset of propositions $\mathbb{P}$ (henceforth referred to as model parameters), where*

$$\mathbb{P} = \{\textit{init-has-f} \mid f \in F\} \cup \{\textit{goal-has-f} \mid f \in F\} \cup \bigcup_{a \in A} \{\textit{a-has-prec-f}, \textit{a-has-add-f}, \textit{a-has-del-f} \mid f \in F\}.$$

*The parameterization function $\delta(\mathcal{M})$ is defined as*

$$\begin{aligned}
\tau_I &= \{\textit{init-has-f} \mid f \in I\} \\
\tau_G &= \{\textit{goal-has-g} \mid g \in G\} \\
\tau_{pre(a)} &= \{\textit{a-has-prec-f} \mid f \in pre(a)\} \\
\tau_{add(a)} &= \{\textit{a-has-add-f} \mid f \in add(a)\} \\
\tau_{del(a)} &= \{\textit{a-has-del-f} \mid f \in del(a)\} \\
\tau_a &= \tau_{pre(a)} \cup \tau_{add(a)} \cup \tau_{del(a)} \\
\tau_A &= \bigcup_{a \in A^{\mathcal{M}}} \tau_a \\
\delta(\mathcal{M}) &= \tau_I \cup \tau_G \cup \tau_A
\end{aligned}$$

We use notation $\delta^{-1}$ to map subsets of $\mathbb{P}$ to possible model. To keep the potential space of model updates finite, we only consider changes that remove components and their parameters from the model representation. We examine model updates that disallow previously applicable plans, in other words, changes that constrain the model. We define a constrained version of a model as follows.

Definition 11. *A model $\mathcal{M}' = \langle F', I', A', G'\rangle$ is a constrained version of a model $\mathcal{M}$, if $\delta(\mathcal{M}') \subseteq \delta(\mathcal{M})$ and there exists no action sequence $\pi$ such that $\Gamma_{\mathcal{M}}(I, \pi) \not\subseteq G$ and $\Gamma_{\mathcal{M}'}(I', \pi) \not\subseteq G'$.*

However, finding a potentially constrained version of a model by looking at potential model changes can be an expensive process. Previous works try to simplify this search by assuming access to a limited set of hardening actions [8, 19, 35]. In most network design settings, the cost of potentially updating a part of the model, hence changing the network, may be under-specified and, in some cases, unspecified. This means we will need to consider the possibility of changing any and all parts of the network (and, by extension, the corresponding model description). Unfortunately, for most realistic networks, the possible space of such changes is too large to be explored. However, we show that by focusing on the problem of identifying constrained versions of the model, we can ignore all possible model updates except the ones that are part of the set, called *constraining changes set* or $\kappa(\mathcal{M}) = \tau_I \cup (\bigcup_{a \in A} \tau_{add(a)})$. Now we demonstrate the effectiveness of $\kappa$ by showing that one can only create a constrained version of a model by using changes that are part of the constraining changes set.

Proposition 1. *If a model $\mathcal{M}' = \langle F', I', A', G'\rangle$ is a constrained version of $\mathcal{M} = \langle F, I, A, G\rangle$ such that, $F' = F$ and $\delta(\mathcal{M}') \subseteq \delta(\mathcal{M})$, then $\delta(\mathcal{M}) \setminus \delta(\mathcal{M}') \subseteq \kappa(\mathcal{M})$.*

This follows directly from the properties of models with positive pre-condition only. There is no case where a previously executable plan will be made inexecutable after removing a pre-condition, delete effect, or goal.

Note that our objective here is not just to find any constrained model but rather one that disallows potential attack paths while still preserving paths to nodes of interest. So, we find such models for both metrics by performing a search over the set of possible updates. We do this by employing a modified form of $A^*$-search , a systematic heuristic search. $A^*$ is guaranteed to return an optimal solution (i.e., the least costly set of model updates) as long as the heuristic is an optimistic estimate of the cost of the remaining part of the solution. Specifically, the $A^*$ search maintains a priority queue that is ordered by the sum of the cost of the current search node and the heuristic value associated with getting to a goal node. In our case, the search node corresponds to a potential model you can obtain by applying the possible changes. The cost relates to the cost of changes already applied, and the heuristic approximates the cost of changes that need to be further applied to reach a model where some pre-specified goal condition related to the metrics is met. At any point in the search, the algorithm expands the node with the smallest possible value. If the selected node is a goal node, the search ends; if not, it adds all successor nodes (in our case, generated by applying available model updates), which are then added to the priority queue. In terms of updates to the search algorithm, we introduce changes that are applicable to both metrics, while still maintaining the guarantees provided by $A^*$. We also propose a new heuristic function that will help speed up the search in any context and focus on model updates that constrain the original model.

For the heuristic, we start by assuming that all model updates have a uniform cost of one. This means that for any set of search nodes that is not the goal node, the heuristic is automatically admissible if the value is less than or equal to one (after all, it would take at least one more model update for it to meet the goal condition). However, setting all nodes to have equal heuristic values does not allow us to find more effective solutions. We would want to set lower heuristic values for more promising nodes. Specifically, for a given search node, we will generate a set of attack paths and see which one eliminates the most attack paths among all the successor nodes. Specifically, we will give a heuristic value proportional to

**ALGORITHM 1:** Heuristic function

**Input** : $\hat{\mathcal{M}}$, plan_list
**Output**: heuristic_value

h ← |plan_list|;
**for** *a_plan* ∈ *plan_list* **do**
  **if** *a_plan is not valid in* $\hat{\mathcal{M}}$ **then**
    *h*− = 1;
  **end**
**end**
heuristic_value ← h / |*plan_list*|;
**return** heuristic_value

**ALGORITHM 2:** Search for non-overlapping set

**Input** : $\mathcal{M} = \langle F, A, I, \mathbb{G}_C, \mathbb{G}_R \rangle$, budget $k$
**Output**: Solution Set $\mathbb{K}$
fringe ← `PriorityQueue()`;
c_list ← ⟨⟩ *(Closed list)*;
cnt ← 0;
solution_list← ⟨⟩ ;
fringe.push({}, 0);
**while** *fringe not empty &* $cnt \leq k$ **do**
  $cnt$+ = 1;
  $\hat{\mathcal{C}} \leftarrow$ fringe.pop();
  $\hat{\mathcal{M}} \leftarrow \delta(\mathcal{M}) \setminus \hat{\mathcal{C}}$ **if** *goal condition met for* $\hat{\mathcal{M}}$ **then**
    solution_list.add($\hat{\mathcal{C}}$);
  **end**
  c_list.add($\hat{\mathcal{C}}$);
  **for** $f \in \kappa(\mathcal{M})$ **do**
    **if** $\{f\} \cup \hat{\mathcal{C}}$ *has no subset in* $\hat{\mathbb{C}}$ *and there exists a connectivity path* **then**
      $\hat{\mathcal{C}}' \leftarrow \{f\} \cup \hat{\mathcal{C}}$;
      $\hat{\mathcal{M}}' \leftarrow \delta(\mathcal{M}) \setminus \hat{\mathcal{C}}'$;
      $\Pi_C \leftarrow$ `FindTopAttackPlans`($\mathcal{M}$)
      fringe.push($\hat{\mathcal{C}}', |\mathcal{C}'| +$ heuristic($\hat{\mathcal{M}}', \Pi_C$));
    **end**
  **end**
**end**

the number of still valid attack paths for each successor. While one would ideally want to consider all attack paths, this may not be computationally feasible as we need to perform this on every node expanded by the search algorithm. Thus, we restrict it to some $K$ attack paths. Algorithm 1 presents a pseudo-code for calculating these heuristic values, and we show the effectiveness of this heuristic through empirical evaluation.

The next optimization we introduce into the algorithm is node pruning. In particular, we want to avoid adding nodes to our priority queue that are guaranteed never to lead to a goal. Note that at this point, we have yet to define our goals precisely. Regardless of the exact form the function related to attack paths takes, in each case, we would want $\mathcal{F}_C^1$ to return a value of one. This means we can prune out any search nodes that cannot lead to models where there are connectivity paths.

Proposition 2. *Let $\mathcal{M}$ be an equivalent representation of* ACG *$\mathcal{G}$, an initial node set $S_N^0$ and a set of connectivity target $S_t^C$, such that there exists no valid connectivity path for the graph, then there exists no constrained version of $\mathcal{M}$, where the corresponding* ACG *model has a valid connectivity path.*

This proposition directly follows from the fact that a constraining model update never adds new plans, and a valid plan must exist for any valid connectivity path. This result tells us that we can ignore any successors of a search node where no valid path to a connectivity target exists.

Now that we have covered two modifications that are designed to exploit the structure of the problem to improve the effectiveness of the search process, we extend $A^*$ to handle one of the shortcomings of the setting, namely missing modification costs. $A^*$ is generally deployed in settings where each step that can be taken from a search node has a well-defined cost. All the theoretical guarantees of $A^*$ are contingent on access to a specified cost function. In our setting, each potential step corresponds to some update in the model, which further corresponds to a change in the underlying network. This might include changes ranging from potentially patching software to potentially removing network connections. Humans are generally known to be bad at approximating costs and utilities in general [6], however, such problems are made worse in settings like network administration, where there are many competing interests. For example, even a small change in updating a software version may involve retraining a number of users.

Given these considerations, it is unlikely that the system typically has access to a fully defined cost function associated with the possible network modifications. As such, the search would not be able to identify a single optimal modification set. Instead, we focus on identifying options that meet the required goal conditions. These options can then be presented to the defender, who can then compare the modifications being proposed by each option and select one they might believe is the most reasonable one. When coming up with these options, our primary focus would be to choose solutions that make use of diverse network modifications. The use of diverse solution sets as a means of addressing incompletely specified cost functions is a strategy that has been shown to be effective in many contexts [13]. In this setting, by choosing diverse modifications, we increase the chance that one of the solutions consists of modifications that the user may deem to be less costly.

In particular, we will come up with a set of updates such that no set of model updates is a subset of another. We will call such a set of model updates a non-overlapping set.

Definition 12. *A set $\mathbb{C} = \{C_1, ..., C_k\}$ is a* non-overlapping set *of model updates, if there exists no $1 \leq j \leq k$ and $j \neq i$, such that $C_i \subseteq C_j$.*

We will generate such a non-overlapping set by extending $A^*$ to identify multiple solutions. Effectively, each time a goal condition is met, it will get added to a set of solutions, and no successor node that is a super-set of an already found solution is added to the search priority queue. Here, we impose a search budget that ensures the search exits after a certain number of search nodes are expanded. Algorithm 2 presents the modified $A^*$ search that incorporates all the changes discussed above.

Proposition 3. *Algorithm 2 will always return a solution set that is non-overlapping.*

This proposition holds true, as the search is always guaranteed to test potential solutions in the order of increasing cardinality. This means we only need to ensure that any future solution is not a superset of a previous solution to ensure that the solution set satisfies Definition 12.

*4.2.1 Goal Condition.* Now, one can adapt Algorithm 2 for each individual metric by changing the goal condition. For the impenetrability metric, the most intuitive condition would be to have both $\mathcal{F}^I_{\mathcal{A}}$ and $\mathcal{F}^{\infty}_{\mathcal{C}}$ return one. To identify such models, we need to check that the model does not return any plan for the attack targets and that there still exists a plan for the connectivity target. This can be extended to multiple initial states and goals either by checking pairwise or by using compilation techniques similar to the one discussed below.

However, for attack difficulty, we have a slightly more complex goal condition. We need to make sure that $\mathcal{F}^{\mathcal{D}}_{\mathcal{A}}$ meets certain threshold. The particularly interesting case here is one where we have multiple initial state sets and multiple targets. This means that we need a way of finding the shortest possible attack path from a set of possible starting states to the set of possible attack targets. We will do so by employing a planning compilation that will automatically identify this path.

Our objective here is to build on our planning model to support finding the shortest possible path from multiple initial state sets and multiple targets. We will represent this updated model as $\mathcal{M}^{\alpha} = \langle F^{\alpha}, A^{\alpha}, I^{\alpha}, G^{\alpha} \rangle$. The new fluents $F^{\alpha}$ is given as $F^{\alpha} = F \cup \{\texttt{init\_change\_mode}\} \cup \{\texttt{act\_mode}\} \cup \{\texttt{goal\_reached}\}$ Here $F^{\alpha}$ includes all the original fluents, plus three new fluents. First, we have the proposition `init_change_mode`, which enables the planner to select the starting initial state set. Next, the fluent `act_mode` indicates that the planner can only select the attack actions, and finally, the fluent `goal_reached` indicates that the intended target has been achieved. Our new set of actions $A^{\alpha}$ is as follows: $A^{\alpha} = A^{O} \cup A^{I} \cup A^{G}$.

This set comprises the following components. Firstly, it includes all the original actions of the attacker, denoted as $A^{O}$, but with an added pre-condition `act_mode`. This pre-condition is necessary to ensure that the attack actions are only executed in the appropriate mode of operation. Furthermore, we introduce a new set of actions, $A^{I}$, specifically designed to transition the state to one of the potential initial states. Another subset of actions, $A^{G}$, is introduced to trigger the proposition `goal_reached` when the goal conditions of the attacker are met. More specifically, for all possible initial states $S_i$ we have corresponding action $a^I_i = \langle pre(a^I_i), add(a^I_i), del(a^I_i) \rangle$, such that

$$pre(a^I_i) = \{\texttt{init\_change\_mode}\},$$
$$add(a^I_i) = \gamma^{\mathcal{M}}(S_i) \cup \{\texttt{act\_mode}\},$$
$$del(a^I_i) = \{\texttt{init\_change\_mode}\}$$

The delete effect in the initial state action ensures that it can only select an initial state once and prevents the planner from possibly creating infeasible paths by adding elements from other initial states. Once the initial state action is executed, the fluent `act_mode` is made true, and now the planner can execute the attack actions.

For all possible reachable attack targets $(h_i, p_i)$, we introduce new actions $a^G_i = \langle pre(a^G_i), add(a^G_i), del(a^G_i) \rangle$ with the corresponding definition

$$pre(a^G) = \gamma^{\mathcal{M}}(h_i, p_i),$$
$$add(a^G) = \{\texttt{goal\_reached}\}, del(a^G) = \{\}$$

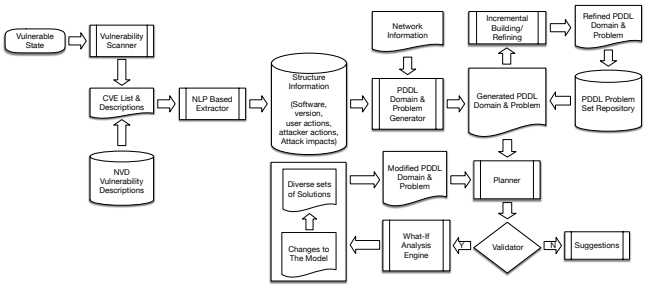



**Figure 1: SPEAR Tool Architecture.**

The initial state consists of only the `init_change_mode` fluent

$$I^{\alpha} = \{\texttt{init\_change\_mode}\}$$

Lastly, the goal becomes

$$G^{\alpha} = \{\texttt{goal\_reached}\}$$

An optimal plan for such a model will be one in which the planner chooses an initial state and goal pair that results in the attack plan with minimal cost.

Proposition 4. *For a given graph $\mathcal{G}$, a set of initial state sets $\mathbb{S}^0_{\mathcal{N}}$ and attack targets, $\mathbb{S}^{\mathcal{A}}_t$, such that a valid attack path exists, the value returned by $\mathcal{F}^{\mathcal{D}}_{\mathcal{A}}$, is equal to the cost of the optimal plan for the corresponding compiled planning model $\mathcal{M}^{\alpha} = \langle F^{\alpha}, A^{\alpha}, I^{\alpha}, G^{\alpha} \rangle$.*

In the compiled model, the planner has the freedom to choose the initial state and the goal state, but it must always select one. Since we are using the optimal planner, it will always choose the initial and goal state that will result in the cheapest possible attack path. Because every valid plan in this domain must include a single initial state selection action and a target selection action. As such, the only place the planner can optimize is in the cost of the attack path. Thus it returns $\min\{|\mathbf{E}| \mid \mathbf{E} \in \mathbb{E}_{\mathcal{A}}\}$.

In this case, the goal check simply involves calling this compilation. If the cost of the minimal plan is greater than or equal to the threshold, then the identified solution is added to the solution list.

## 5 The SPEAR Toolset

The overview of the SPEAR tool is illustrated in Figure 1. SPEAR is integrated with an off-the-shelf network vulnerability scanner that generates a list of vulnerabilities and their descriptions.

Figure 2 shows the test network we used in this study. We created the test network using various virtual environments and tools to simulate real-world scenarios. This setup included virtual machines, network configurations, and firewall rules. The network is divided into two subnetworks: Sub-Network 1 (demilitarized zone) that contains DNS, Mail & Web Server, and Sub-Network 2 (trusted zone) that contains FTP, Database & Admin Server running various services. TABLE 3 (in Appendix B) lists the vulnerabilities present in this network. Before SPEAR is used, we gather information on network topology & connectivity, vulnerabilities, and services. SPEAR will create the *ACG* based on this information. A visual representation of the *ACG* is shown in Figure 6 in Appendix A.

We collected detailed system and network data, categorized as host-level and connectivity attributes. These were transformed

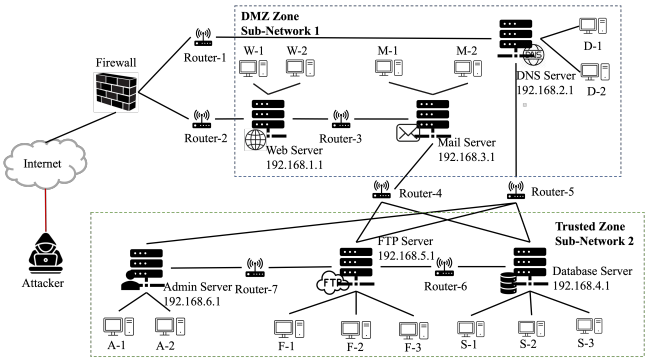



**Figure 2: Topology of Test Network.**

into a formal representation using a domain and problem generator, enabling automated planning. Once generated, both the domain and problem files are introduced to a *Validator*. This *Validator* scrutinizes the plan for a designated goal. A valid plan signifies the existence of a pathway that an attacker could exploit to compromise the target or achieve a specific goal state. If such pathways or plans are discerned, the domain and problem files are then relayed to a *what-if-analysis* engine. For *impenetrability metric*, the analysis engine conducts an exhaustive exploration, by constrained model updates, to find all potential sets of changes that would obstruct any plans leading to the goal state. Each derived set of changes culminates in a solution, which translates into a revised domain and problem file. For each unique set of changes, or, in other words, for each distinct solution, the *Planner* indicates that the original plans are invalid. Delving deeper, the *Validator* identifies and delineates the specific initial states where the actions prove ineffective. Lastly, the *Validator* genrate suggestions which include both the failed plan and its associated unsuccessful actions, to providing users with insights into the modifications required within network. For, *attack difficulty metric*, we are finding the set of changes for which the minimum cost for the attacker to reach the target node increases in the *Planner*. Considering that the weakness of a network depends on its most vulnerable attack path, this analysis significantly increases the difficulty for attackers within a specific network.

Here we are introducing *human-in-the-loop* concepts where the administrator would have all possible sets of diverse changes and their corresponding solutions. They can choose any possible change. Also, if there are any new modifications in the network, we introduced continuous building and refining options for updating the previous model copies.

## 6 What-if Analysis Using SPEAR

In this section, we will discuss how our tool can be used in the context of the network presented in Figure 2 to perform *what-if analysis*. The reader may want to refer to the corresponding *ACG* shown in Figure 6 in Appendix A.

### 6.1 Impenetrability Analysis

Our first analysis is based on *impenetrability metric* (see Definition 8). Typically, the system administrator starts by selecting the network node they would like to focus on, say the Admin Server. Since SPEAR models the attack-connectivity graph as a planning model, we can readily use existing solvers to present the users with a set of diverse attack vectors of different qualitative characteristics (via forbid-iterative). In this case, we found 21 different sets of attack paths to compromise the Admin Server.

However, the goal of our tool is to go beyond such shallow analysis and help the user directly identify ways in which attack paths to the node can be disrupted while ensuring that the node is still accessible from other nodes in the network. Using our model space search technique, SPEAR identifies unique ways in which the security stances of the node in question may be hardened. Each of these different updates may pose unique challenges to implement, which may be hard to implement but easier for a domain expert to assess. In this case, the search identifies a set of diverse possible changes shown in Figure 3.

```
; SPEAR Tool Output:
Changes -> {'has-initial-state-admin-driver-1 gmetad-slash-server-
      dot-c', 'has-initial-state-admin-driver-2 mpm-event-worker-or
      -prefork'}
Changes -> {'has-initial-state-adminserver-system-1-config ganglia
      -3-dot-1-dot-1', 'has-initial-state-admin-driver-2 mpm-event-
      worker-or-prefork'}
Changes -> {'has-initial-state-vul-admin-function-1 process_path-
      function', 'has-initial-state-admin-driver-2 mpm-event-worker
      -or-prefork'}
Changes -> {'has-initial-state-adminserver-system-2-config apache-
      http-server-2-dot-4-dot-17-to-2-dot-4-dot-38', 'has-initial-
      state-admin-driver-1 gmetad-slash-server-dot-c'}
```

**Figure 3: Changes suggested by SPEAR Tool.**

Note that, given the level of abstraction used in the modeling, the system is able to give suggestions at the level of potential changes in network topology or specific vulnerabilities. The question of how to implement these changes is left to the administrator. Once presented with the options, the administrator is free to consider each option and validate it by implementing those network changes.

If the administrator selects 2$^{nd}$ change (transalted in natural language by SPEAR ) – *Upgrade the ganglia software's version anything other than v3.1.1 to a more recent version, and modify the MPM event worker or prefork driver* – shown in Fig. 3, then the above attack plans would fail. These case study proves that the *impenetrability metric*, $\mathcal{F}^{\mathcal{I}}_{\mathcal{A}}(\mathbb{E}_{\mathcal{A}}) = 1$ as there are no valid attack paths ($|\mathbb{E}_{\mathcal{A}}| = 0$) to compromise Admin Server. The administrator is now free to investigate other potential attack paths or even sketch a potential attack path and see how it might fail in the context of a given model update. The number of changes is not limited to 4, as the planner can generate more diverse sets of suggested changes for other nodes that could potentially disrupt the attacker from reaching the Admin Server. By opting for any of these model changes, all the attack paths targeting Admin Server are now invalid but the connectivity paths or services to that Admin Server are still valid ($\mathbb{E}_C$ is valid), so the connectivity function, $\mathcal{F}^1_C(\mathbb{E}_C) = 1$. Figure 4 shows a valid plan to reach the Admin Server when the model changes are applied and there exists no plan to compromise the Admin Server.

When conducting our *what-if analysis*, we factored in the "human-in-the-loop" perspective. For the sake of simplification, we initially assigned a unit cost for each change. However, in real-world applications, this unit cost might vary based on the nature and impact of the change. For instance, while the tool might view all sets of changes as having the same unit costs, the actual implications for

```
(has_access_to_mail_server web-server mail-server internet gateway
      -192-168-3-1 p-25 smtp)
(has_access_to_ftp_server_via_mail_server   mail-server ftp-server
      internet gateway-192-168-5-1 p-20 tcp)
(has_access_to_admin_server_via_ftp_server ftp-server admin-server
      internet gateway-192-168-4-1 p-1311 https)
```

**Figure 4: Plan (Connectivity Path $\mathbb{E}_C$) to reach Admin Server.**

an organization or administrator might differ significantly. For example, for changes to *MPM event worker or prefork* and *Ganglia version 3.1.1*, any changes to the former might be perceived as more cumbersome and thus more costly than updating the latter to a newer version. Consequently, the administrator might prefer to implement the 2nd set of changes over the 3rd set, even if its unit cost is the same. This decision underscores the importance of accounting for the nuanced and often unpredictable nature of human decision-making in our analysis.

## 6.2 Attack Difficulty Analysis

The other notable application of the SPEAR tool is its capability to augment the minimum cost associated with all possible initial and target states from an attacker's perspective. In this analysis, we are focusing on metric 2 or *attack difficulty metric* (Definition 9). Considering that a network's robustness is contingent upon its weakest attack path, this analysis effectively elevates the challenge for attackers in a given network. Suppose an administrator aims to escalate the minimum cost of the attack path under various potential scenarios, which constitute possible initial states for attackers:

(1) Attacker has access to the DNS server's vulnerabilities.
(2) Attacker has access to the Mail server's vulnerabilities.
(3) Attacker has access to SQL server's vulnerabilities.

Based on these initial states we found that the attacker can compromise the Admin Server, provided the attacker's initial state is established subsequent to the exploitation of CVE-2017-14491 (i.e., attack path $\mathbb{E}_{\mathcal{A}}$) with an *attack difficulty metric*, $\mathcal{F}_{\mathcal{A}}^{\mathcal{D}}(\mathbb{E}_{\mathcal{A}}) = 3$. Now, let us assume the administrator seeks to identify model updates that would result in a minimum increase of 2 unit costs.

```
; SPEAR Tool Output:
Changes -> {'has-initial-state-has-adminserver-system-2-config
      apache-http-server-2-dot-4-dot-17-to-2-dot-4-dot-38'}
Changes -> {'has-initial-state-has-admin-driver-2 mpm-event-worker
      -or-prefork'}
Changes -> {'has-initial-state-has-dns-driver-2 request-handling',
      'has-initial-state-has-dnsserver-system-1-config dnsmasq-
      before-2-dot-78'}
Changes -> {'has-initial-state-has-dnsserver-system-1-config
      dnsmasq-before-2-dot-78', 'has-initial-state-has-dnsserver-
      system-2-config windows-dns-server'}
```

**Figure 5: Changes suggested by SPEAR to increase the minimum cost of the given initial states and all possible targets.**

By employing our model space search technique, the administrator can select the modification depicted in Figure 5 to guarantee an increase (at least 2 unit cost) in the cost of compromising one of the possible targets. Specifically, if an administrator implements the first set of suggestion shown in Figure 5, the minimum cost of the attack path ($\mathbb{E}_{\mathcal{A}}$) increases, and the new *attack difficulty metric*, $\mathcal{F}_{\mathcal{A}}^{\mathcal{D}}(\mathbb{E}_{\mathcal{A}}) = 5$. Note, in this scenario, the minimum cost of the attack path is now associated with compromising the Mail Server following the implementation of the model updates. This method ensures that all previously optimal attack plans become invalid due to the increased cost. Consequently, any new plans developed post-update will incur a cost higher than that of the earlier plans.

# 7 Empirical Evaluation

Our primary focus is to evaluate the performance characteristics of SPEAR changed with respect to the complexity of network and attacker goals. We ran our experiment on a 3.1 GHz Dual-Core Intel Core i5 processor, with 8 GB 2133 MHz, and LPDDR3 memory.

**Table 1: Comparison of Impenetrability (M1) and Attack Difficulty (M2) Analysis, with and without Heuristics.**

| # Nodes | Time (s) | | | |
|---|---|---|---|---|
| | M1 | M1 + Heuristic | M2 | M2 + Heuristic |
| Admin | 20.96 | 23.54 | 1540.27 | 1493.88 |
| DNS | 77.75 | 29.14 | 447.39 | 843.68 |
| FTP | 169.75 | 246.86 | 20335.41 | 31762.01 |
| Mail | 9.94 | 14.79 | 145.18 | 492.45 |
| Database | 131.93 | 260.23 | 300.13 | 981.25 |
| Web | 7.43 | 6.96 | 103.41 | 509.33 |

We first investigate how the performance of the SPEAR tool changes with respect to the computation time as we vary the target (goal) node. The purpose of this experiment is to understand the tool's performance for Impenetrability and Attack Difficulty Analysis. TABLE 1 presents the time required to identify two distinct solutions for each potential goal or attacker's target. For this analysis, the value of $\alpha$ is set to 2 (defining the threshold for considering an improvement in robustness; a solution is deemed to enhance this security only if it increases by at least 2), enabling the examination of solutions under metric 2 (*attack difficulty metric*).

In our example, solutions are simple, usually requiring only one or two steps. Computing multiple plans takes longer, and in some cases, skipping the heuristic and selecting updates randomly led to faster results. Thus, the time saved by the heuristic doesn't always outweigh the cost of extra plan calculations. To explore the effectiveness of our approach in more complex scenarios, we evaluated it by varying the number of nodes and vulnerabilities.

**Table 2: Comparison of Time Required for Various Nodes.**

| # Nodes | Time (s) | | | |
|---|---|---|---|---|
| | M1 | M1+Heuristic | M2 | M2+Heuristic |
| 10 | 260.78 ± 149.72 | 213.55 ± 129.21 | 97.2 ± 92.07 | 88.67 ± 97.22 |
| 15 | 348.57 ± 299.74 | 260.01 ± 251.46 | 526.36 ± 479.95 | 289.15 ± 345.79 |
| 20 | 4389.53 ± 3210.5 | 2664.91 ± 1616.98 | 673.04 ± 690.33 | 216.13 ± 147.66 |
| 25 | 1293.94 ± 1224.48 | 244.68 ± 147.07 | 8071.04 ± 12964.31 | 3475.81 ± 4387.2 |
| 30 | 3266.08 ± 2712.53 | 1680.93 ± 2123.6 | 7404.54 ± 11908.29 | 3756.9 ± 7200.47 |

The second factor we investigate is the computational characteristics of our proposed algorithm when we increased the number of nodes in the network to address the scalability issues in network security. We generated directed, scale-free network topologies using the Barabási-Albert model to simulate realistic network environments for *what-if analysis*, as this model reflects the power-law structure commonly found in real-world networks. We varied the total number of nodes (10,15,20,25,30) and vulnerabilities associated

with each node, and each new node added during the network's construction connects to 2 existing nodes.

TABLE 2 displays the time taken (in seconds, along with their standard deviations) to find two distinct solutions for metrics 1 and 2, we evaluated results with and without using heuristics, averaging across varying numbers of vulnerabilities per node group. Specifically, each node group is analyzed across four problem instances with vulnerabilities representing 20%, 40%, 60%, and 80% of the total number of nodes, with vulnerabilities randomly distributed. For example, a graph with 10 nodes includes problem instances with 2, 4, 6, and 8 randomly distributed vulnerabilities. While metric 1 uses a fixed initial and goal node for each problem instance, metric 2 employs three randomly selected initial nodes and three randomly chosen goal nodes, and the $\alpha$ value for metric 2 is set to 2. As observed from the table, using heuristics reduces the time required to find solutions for both metrics across various node counts.

The search utilizes modified $A^*$ to identify model updates, it is crucial to prioritize which updates to apply first. The heuristic we use effectively manages this prioritization. Specifically, for both metrics 1 and 2, each current model update begins with the calculation of three different plans, based on which the heuristic evaluations are made (as detailed in Algorithm 2). This process provides insights into how close the subsequent updates are to achieving the desired final outcomes.

## 8 Related Work

Attack graphs (and variants) have been variously studied by the security research community for modeling and analyzing security postures of networked systems and drive cyber risk management (see for example, [24, 26–28, 30]). The main efforts have been in addressing issues such as automatically generating attack graphs and identifying attack paths [18], visualization of security postures [17], attack graphs as analysis tool [38], and for security hardening [9, 10, 25]. Poolsappasit et al. introduced Bayesian Attack Graphs (BAG) [29] for risk evaluation and mitigation planning. Beckers et al. combined high-level attack tree analysis and low-level attack graph analysis to compute the probability of successful attacks [3]. This process was extended in Beckers et al. by considering the effects of social engineering threats [4].

AI-Planning tools have been introduced in the cyber-security applications which use PDDL to generate attack graph [2, 5, 14, 31, 36]. Ghosh et al. proposed Planner [14], a specialized search algorithm from the field of AI, to generate scalable and time-efficient representations of attack graphs. Tiwary et al. introduced a methodology and a tool called PDDLAssistant [36] to facilitate the incremental development of PDDL representations for cyber-security domains. This process is further enhanced in Bezawada et. al. where they introduced AGBuilder [5] to address the limitations of scalable attack graph generation and representation.

Attack graphs have often proven to be versatile tools for enhancing the analysis and understanding of complex security vulnerabilities across various systems [37]. Albanese et al. provided polynomial time algorithms [1] that evaluated k-zero-day safety in large networks. However, this work is not suited for what-if analysis. Miller et al. introduced GRASP [22], leveraging machine learning to optimize attack path identification, which significantly speeds up computational times without compromising on accuracy. Complementing this, Milani et al. [21] delve into the strategic use of deception in security games. However, a limitation was the need for additional memory page transfers among search agents and coordination among them. Our proposed tool SPEAR is able to handle the complexity and scalability issues as we are using an AI planner with heuristics.

In the field of network hardening, researchers have made significant contributions, building upon each other's work to advance the understanding and application of this approach [9, 10]. Dewri et al. [9] introduced the notion of optimal security hardening by developing attack tree cost models for network attack and defense. Later, they extended this work by considering the attacker's perspective in network hardening [10]. Roberts et al. proposed an alternative approach to home computer security software using a software agent [32] that could assist users in managing their security risks. But, traditional attack graph analysis approach treats network hardening as a multi-objective optimization problem, focuses on producing an "optimal" solution based on predefined criteria. In contrast, SPEAR enables what-if analysis, allowing sys-admin to explore multiple scenarios and choose the most desirable course of action based on cost, availability of resources, human intuition, or other constraints. Sometimes an optimal solution may not be a viable solution for a particular scenario. The key difference is that, SPEAR could incorporate human decision-making (via what-if analysis) rather than replacing it with automated optimization.

## 9 Conclusion and Future Work

In this work, we have presented a method for integrating AI planning techniques with attack graphs to enhance network security. Our contributions involve modeling vulnerabilities and their dependencies as well as network connectivity as a hypergraph called Attack-Connectivity Graph (*ACG*) using the PDDL. *ACG* is non-monotonic, unlike most other similar models in the prior art. We develop the SPEAR toolset for evaluating Attack-Connectivity Graph creating unsolvable plans for proposed changes. The tool allows a human-in-the-loop interaction. These features contribute synergistically to allow the administrator to perform what-if analysis and also to reason about different attacker motives. The logic and reasoning framework of SPEAR closely matches how human users reason about actions, and thus the suggestions generated are intuitive and easy to understand. Our future work includes improving the framework to allow for cost-benefit analysis for network hardening. We also plan to do usability studies to understand how suggestions provided by SPEAR are perceived by domain experts.

## Acknowledgment

This work was partially supported by the U.S. National Science Foundation under Grant No. 1822118 and 2226232, Award Numbers DMS 2123761, by the industry member partners of the NSF IUCRC Center for Cyber Security Analytics and Automation -– AMI, NewPush, Cyber Risk Research, NIST and ARL -– by the State of Colorado (grant #SB 18-086) and by the authors' institutions. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, or other organizations and agencies.

## References


[1] Massimiliano Albanese, Sushil Jajodia, Anoop Singhal, and Lingyu Wang. 2013. An Efficient Approach to Assessing the Risk of Zero-Day Vulnerabilities. In *Proceedings of 2013 International Conference on Security and Cryptography (SECRYPT)*, Pierangela Samarati (Ed.). Reykjavik, Iceland, 1–12.

[2] Shadaab Kawnain Bashir, Rakesh Podder, Sarath Sreedharan, Indrakshi Ray, and Indrajit Ray. 2024. Resiliency Graphs: Modelling the Interplay between Cyber Attacks and System Failures through AI Planning. In *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*. IEEE, 292–302.

[3] Kristian Beckers, Maritta Heisel, Leanid Krautsevich, Fabio Martinelli, Rene Meis, and Artsiom Yautsiukhin. 2014. Determining the Probability of Smart Grid Attacks by Combining Attack Tree and Attack Graph Analysis. In *Proceedings of Smart Grid Security: Second International Workshop, SmartGridSec 2014 (Lecture Notes in Computer Science, Vol. 8448)*, Jorge Cuellar (Ed.). Springer, Munich, Germany, 30–47.

[4] Kristian Beckers, Leanid Krautsevich, and Artsiom Yautsiukhin. 2015. Analysis of Social Engineering Threats with Attack Graphs. In *Proceedings of Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance: 9th International Workshop, DPM 2014, 7th International Workshop, SETOP 2014, and 3rd International Workshop, QASA 2014 (Lecture Notes in Computer Science, Vol. 8872)*. Springer, Wroclaw, Poland, 216–232.

[5] Bruhadeshwar Bezawada, Indrajit Ray, and Kushagra Tiwary. 2019. AGBuilder: An AI Tool for Automated Attack Graph Building, Analysis, and Refinement. In *Data and Applications Security and Privacy XXXIII: 33rd Annual IFIP WG 11.3 Conference, DBSec 2019, Charleston, SC, USA, July 15–17, 2019, Proceedings 33*. Springer, 23–42.

[6] Serena Booth, W Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. 2023. The Perils of Trial-and-Error Reward Design: Misdesign through Overfitting and Invalid Task Specifications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 5920–5929.

[7] Tom Bylander. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence* 69, 1-2 (1994), 165–204.

[8] Turgay Caglar, Sirine Belhaj, Tathagata Chakraborty, Michael Katz, and Sarath Sreedharan. 2024. Can LLMs Fix Issues with Reasoning Models? Towards More Likely Models for AI Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 20061–20069.

[9] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. 2007. Optimal Security Hardening using Multi-Objective Optimization on Attack Tree Models of Networks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*. 204–213.

[10] Rinku Dewri, Indrajit Ray, Nayot Poolsappasit, and Darrell Whitley. 2012. Optimal Security Hardening on Attack Tree Models of Networks: A Cost-Benefit Analysis. *International Journal of Information Security* 11, 3 (June 2012), 167–188. doi:10.1007/s10207-012-0160-y

[11] Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, Christopher Kiekintveld, and Michal Pěchouček. 2019. Hardening Networks Against Strategic Attackers using Attack Graph Games. *Computers & Security* 87 (2019), 101578.

[12] Erella Eisenstadt and Amiram Moshaiov. 2016. Novel solution approach for multi-objective attack-defense cyber games with unknown utilities of the opponent. *IEEE Transactions on Emerging Topics in Computational Intelligence* 1, 1 (2016), 16–26.

[13] Mahsa Ghasemi, Evan Scope Crafts, Bo Zhao, and Ufuk Topcu. 2021. Multiple Plans are Better than One: Diverse Stochastic Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 31. 140–148.

[14] Nirnay Ghosh and Soumya K Ghosh. 2012. A Planner-Based Approach to Generate and Analyze Minimal attack Graph. *Applied Intelligence* 36 (2012), 369–390.

[15] Jörg Hoffmann. 2015. Simulated Penetration Testing: From" Dijkstra" to" Turing Test++". In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 25. 364–372.

[16] Jörg Hoffmann and Bernhard Nebel. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *Journal of Artificial Intelligence Research* 14 (2001), 253–302.

[17] Mariam Ibrahim, Qays Al-Hindawi, Ruba Elhafiz, Ahmad Alsheikh, and Omar Alquq. 2019. Attack Graph Implementation and Visualization for Cyber Physical Systems. *Processes* 8, 1 (2019), 12.

[18] Kerem Kaynar and Fikret Sivrikaya. 2015. Distributed Attack Graph Generation. *IEEE Transactions on Dependable and Secure Computing* 13, 5 (2015), 519–532.

[19] Sarah Keren, Avigdor Gal, and Erez Karpas. 2019. Goal Recognition Design in Deterministic Environments. *Journal of Artificial Intelligence Research* 65 (2019), 209–269.

[20] Bertram F Malle. 2006. *How the Mind Explains Behavior: Folk Explanations, Meaning, and Social Interaction*. MIT press.

[21] Stephanie Milani, Weiran Shen, Kevin S Chan, Sridhar Venkatesan, Nandi O Leslie, Charles Kamhoua, and Fei Fang. 2020. Harnessing the Power of Deception in Attack Graph-Based Security Games. In *Decision and Game Theory for Security: 11th International Conference, GameSec 2020, College Park, MD, USA, October 28–30, 2020, Proceedings 11*. Springer, 147–167.

[22] Zohair Shafi Miller, A Benjamin, Ayan Chatterjee, Tina Eliassi-Rad, and Rajmonda S Caceres. 2023. GRASP: Accelerating Shortest Path Attacks via Graph Attention. *arXiv preprint arXiv:2310.07980* (2023).

[23] Thanh H Nguyen, Mason Wright, Michael P Wellman, and Satinder Baveja. 2017. Multi-Stage Attack Graph Security Games: Heuristic Strategies, with Empirical Game-Theoretic Analysis. In *Proceedings of the 2017 Workshop on Moving Target Defense*. 87–97.

[24] Steven Noel and Sushil Jajodia. 2009. Advanced Vulnerability Analysis and Intrusion Detection through Predictive Attack Graphs. *Critical Issues in C4I, Armed Forces Communications and Electronics Association (AFCEA) Solutions Series. International Journal of Command and Control* (2009).

[25] Steven Noel and Sushil Jajodia. 2014. Metrics Suite for Network Attack Graph Analytics. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*. 5–8.

[26] S. Noel, Sushil Jajodia, B. O'Berry, and M. Jacobs. 2003. Efficient Minimum-Cost Network Hardening via Exploit Dependency Graphs. In *Proceedings of 19th Annual Computer Security Applications Conference*, Frances Titsworth (Ed.). IEEE, Las Vegas, NV, USA, 86–95. doi:10.1109/CSAC.2003.1254313

[27] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. 2005. MulVAL: A Logic-Based Network Security Analyzer. In *Proceedings of the 14th Conference on USENIX Security Symposium* (Baltimore, MD) *(SSYM'05, Vol. 8)*. USENIX Association, Baltimore, MD, USA, 113–128.

[28] Cynthia Phillips and Laura Painton Swiler. 1998. A Graph-Based System for Network-Vulnerability Analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms* (Charlottesville, Virginia, USA) *(NSPW '98)*. Association for Computing Machinery, New York, NY, USA, 71–79. doi:10.1145/310889.310919

[29] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. 2011. Dynamic Security Risk Management Using Bayesian Attack Graphs. *IEEE Transactions on Dependable and Secure Computing* 9, 1 (2011), 61–74.

[30] Indrajit Ray and Nayot Poolsapassit. 2005. Using Attack Trees to Identify Malicious Attacks from Authorized Insiders. In *Proceedings of Computer Security – ESORICS 2005*, Sabrina de Capitani di Vimercati, Paul Syverson, and Dieter Gollmann (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 231–246.

[31] Indrajit Ray, Sarath Sreedharan, Rakesh Podder, Shadaab Kawnain Bashir, and Indrakshi Ray. 2023. Explainable AI for Prioritizing and Deploying Defenses for Cyber-Physical System Resiliency. In *2023 5th IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. 184–192. doi:10.1109/TPS-ISA58951.2023.00032

[32] Mark Roberts, Adele E Howe, Indrajit Ray, and Malgorzata Urbanska. 2012. Using Planning for a Personalized Security Agent. In *Proceedings of Workshop on Problem Solving using Classical Planners at 26th AAAI Conf. on Artificial Intelligence*. Toronto, Ontario, Canada.

[33] B. Schneier. 1999. Attack Trees: Modeling Security Threats. *Dr. Dobb's Journal of Software Tools 24* 12 (1999), 21–29.

[34] Sarath Sreedharan, Pascal Bercher, and Subbarao Kambhampati. 2022. On the Computational Complexity of Model Reconciliations. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence and the 25th European Conference on Artificial Intelligence, IJCAI-ECAI (IJCAI-22)*. Messe Wien, Vienna, Austria, 4657–4664.

[35] Sarath Sreedharan, Tathagata Chakraborti, and Subbarao Kambhampati. 2021. Foundations of Explanations as Model Reconciliation. *Artificial Intelligence* 301, 103558 (2021).

[36] Kushagra Tiwary, Sachini Weerawardhana, Indrajit Ray, and Adele Howe. 2017. PDDLAssistant: A Tool for Assisting Construction and Maintenance of Attack Graphs Using PDDL. In *in Proceedings of the ACM Conference on Computer and Communications Security 2017 (CCS 2017)*. Dallas, USA.

[37] Simon Unger, Ektor Arzoglou, Markus Heinrich, Dirk Scheuermann, and Stefan Katzenbeisser. 2023. Risk Assessment Graphs: Utilizing Attack Graphs for Risk Assessment. *arXiv preprint arXiv:2307.14114* (2023).

[38] Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. 2021. Structural Attack Against Graph Based Android Malware Detection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3218–3235.


## Appendix A: Visual Representation of Attack-Connectivity Graph

Figure 6 provides a visual representation of the *ACG* for the test network used in this work and described earlier in Section 5. In *ACG* we have two types of directed edges. The normal directed edges indicate that the attacker exploits a particular vulnerability to compromise a node. The node/host remains functional after the attack, allowing the attacker to use it as a reference to compromise additional nodes. But the directed edges with a cross represent that

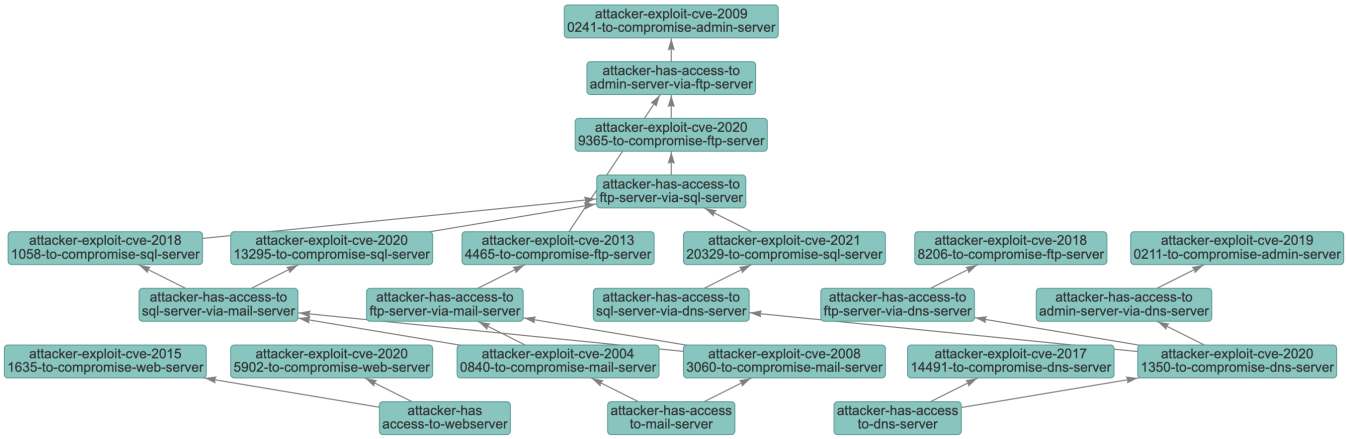



**Figure 6: Attack-Connectivity Graph of Test Network.**

an attacker exploits a particular vulnerability to compromise a node, rendering it nonfunctional post-attack. For example, an attacker might exploit a heap-based buffer overflow vulnerability in a DNS Server machine (D-1) to launch a DoS attack, which then causes the machine to crash and become unusable for further attacks.

## Appendix B: Vulnerabilities in the network

TABLE 3 represents the host/node in the network, its IP address. W-1, W-2, . . . F-3 represent various services running on each host, and their vulnerabilities along with the CVEs associated with each host.

**Table 3: Vulnerabilities (CVE ID) on each server.**

| Host/Node | Services: Vulnerabilities | CVE# |
|---|---|---|
| Web Server (192.168.1.1) | W-1: Remote code execution in HTTP.sys | CVE-2015-1635 |
| | W-2: Remote code execution | CVE-2020-5902 |
| DNS Server (192.168.2.1) | D-1: Heap based buffer-overflow | CVE-2019-0211 |
| | D-2: Remote code execution | CVE-2020-1350 |
| Mail Server (192.168.3.1) | M-1: Arbitrary code execution | CVE-2004-0840 |
| | M-2: Sensitive information disclosure via malformed input and invalid session ID | CVE-2008-3060 |
| Database Server (192.168.4.1) | S-1: Code execution with superuser permissions | CVE-2018-1058 |
| | S-2: SSRF via malicious dockerd server | CVE-2020-13295 |
| | S-3: Improper validation of cstrings | CVE-2021-20329 |
| Admin Server (192.168.5.1) | A-1: Stack based buffer-overflow | CVE-2009-0241 |
| | A-2: Arbitrary code execution via scoreboard manipulation | CVE-2022-37835 |
| FTP Server (192.168.6.1) | F-1: Unrestricted file upload | CVE-2013-4465 |
| | F-2: Out-of-bounds (OOB) read in function | CVE-2020-9365 |
| | F-3: Windows FTP Server DoS | CVE-2018-8206 |