Efficient Attack Countermeasure Selection Accounting for Recovery and Action Costs

Jukka Soikkeli
Imperial College London
United Kingdom
j.soikkeli@imperial.ac.uk

Luis Muñoz-González
Imperial College London
United Kingdom
l.munoz@imperial.ac.uk

Emil Lupu
Imperial College London
United Kingdom
e.c.lupu@imperial.ac.uk

ABSTRACT

The losses arising from a system being hit by cyber attacks can be staggeringly high, but defending against such attacks can also be costly. This work proposes an attack countermeasure selection approach based on cost impact analysis that takes into account the impacts of actions by both the attacker and the defender.

We consider a networked system providing services whose functionality depends on other components in the network. We model the costs and losses to service availability from compromises and defensive actions to the components, and show that while containment of the attack can be an effective defense, it may be more cost-efficient to allow parts of the attack to continue further whilst focusing on recovering services to a functional state. Based on this insight, we build a countermeasure selection method that chooses the most cost-effective action based on its impact on expected losses and costs over a given time horizon. Our method is evaluated using simulations in synthetic graphs representing network dependencies and vulnerabilities, and performs well in comparison to alternatives.

CCS CONCEPTS

• **Security and privacy → Vulnerability management**; **Network security**; • **Networks → Network reliability**;

KEYWORDS

Countermeasure selection, Cyber resilience, Impact analysis.

1 INTRODUCTION

Organisations providing services across the Internet or otherwise connected to an external network face the possibility of cyber-attacks against their systems. Consequently, such organisations should invest in cyber security to protect their services, both to lower the risk of attacks, and to reduce the impact when they occur. Determining the desired level for this investment, and its correct application, is not as straightforward as attempting to secure everything fully. The literature on cyber security investment has shown that a company should never invest to the extent as to cover all potential vulnerabilities or weaknesses [9], and that a company should retain some part of the security investment budget for when an attack event has taken place [5, 10]. Furthermore, given the existence of unknown vulnerabilities and exploits [16], not all eventualities can be prepared against. Additionally, security actions can cause limitations to availability, e.g. reductions to communication between systems or users, or loss of compatibility between software applications due to patching. Combined, these observations lead to a concern over the cyber-resilience of the system consisting of services the organisation provides and the related network, i.e. the ability of the system to withstand and recover from cyber attacks.

Given the possibility that an attack against an organisation's system will occur, some cyber-security investment should be allocated into improving the ability of the system to cope during an attack, and recover to normal functionality. The actions organisations take to mitigate the effects of attacks is the subject of the literature in attack countermeasure selection, reviewed by [15]. In contrast to existing work on countermeasure selection, we take a longer term view by focusing on network resilience, forcing the countermeasure selection to consider impacts over time, and recovery dynamics.

We propose a method for countermeasure selection based on cost impact assessment of both attacker and defender actions, in a medium-to-long-run setting including recovery dynamics. The impacts of actions are estimated using an approach that builds on the attack impact analysis approach by [1, 2], which uses attack graphs (AGs) and dependency graphs (DGs) to estimate attack impact. Our work adds a cost structure, specifying costs for loss of node availability, the countermeasures, and recovery actions. This extends the impact assessment beyond attack impact alone, including the cost impact of defender's actions alongside that from attack steps. The structure enables dynamic semi-automatic countermeasure selection based on the overall costs of alternative defender actions. This allows our method to make more nuanced countermeasure decisions, balancing attack containment and focus on recovery according to the situation to yield efficient countermeasure strategies.

Our main contribution on countermeasure selection, beyond the use of our novel impact assessment, is a method to evaluate the effectiveness of each countermeasure based on its expected impact on the system. This involves the effects of the countermeasure on the possible evolution of the attack, and on the network's service provision, both immediately and in a longer term. We achieve this by using the network attack graph to form expectations of possible attack paths and their likelihoods, and employing the impact assessment method to estimate the cost impact of the different states the system could enter within a few steps from the current state.

We test our method by simulations on a small sample graph and on synthetic graphs, against two alternative methods. The results suggest that our method provides more cost-efficient countermeasure selection than the alternatives tested, especially when attacks are detected with a delay. As our method builds on the attack impact assessment by [1, 2], we made comparisons to an alternative that employs the principles of their attack impact assessment without our extensions. The comparisons show that, for countermeasure selection, our extensions to their analysis add considerable improvements to cost effectiveness and average service performance.

In summary, our contributions include: 1. Extending the attack impact assessment model by [1, 2] into a more general model for impact assessment including defender actions, via modelling service losses and the costs of the countermeasures, and adding a change to enable the modelling of recovery, which the original formulation does not support; 2. Introducing an approach to countermeasure selection based on estimating the expected impacts of actions; 3. Showing that considering recovery and the costs of actions over time can yield a more efficient countermeasure selection.

The paper is structured as follows: Sec. 2 discusses related work; Sec. 3 introduces our impact analysis model, including concepts and definitions from [1, 2] which we build upon; Sec. 4 introduces our countermeasure selection approach; Sec. 5 evaluates the method against two alternatives, using simulations; and Sec. 6 concludes.

2 RELATED WORK

In a general sense, **resilience** is the ability of a system (an organism, a network, a country) to withstand and recover from adverse events such as natural disasters, epidemics, system faults or cyber attacks. Due to the wide use of the term in different fields of study, various definitions for the term have been used [11]. In the definition we shall use, the defining characteristic of resilience is the *recovery and adaptation* exhibited by a system during and after an adverse event. The systems ability to simply weather an event, which we call *robustness* after [8], covers only part of resilience in our definition.

A key work in the field of resilience measurement, Bruneau et al. [3] introduced a way to measure resilience based on a system performance curve. Their method calculates resilience loss, the performance lost due to lack of resilience. This is the difference between the level of a performance metric and its optimal value over time, from the start of the disruption to when performance has recovered fully. While other measures have been proposed, such as a method combining two metrics for measuring the resilience of a backbone network by Sterbenz et al. [19, 20], approaches based on performance curves akin to [3] have become common in the literature for quantifying system resilience, used by e.g. [8, 22].

Our work differs from most of the existing literature on resilience by focusing on the actions and investment choices during an ongoing event (attack) and the recovery phase, instead of preparatory planning and capability investment. This focus is intended to address the evolving nature of systems, and adaptation to conditions such as loss of confidentiality or network unavailability. Most cyber resilience works have focused on the planning and design stage, such as [8, 18, 20]. Additionally, papers considering reactive response and recovery apply to narrow settings unrelated to our work. For example, the approaches by [4, 22] apply to settings where a control action to correct for a deviation from desired performance is easy to determine in advance, and to apply automatically.

In the cyber security literature, **countermeasure selection** refers to approaches to choose actions to counter security events (cyber attacks) [15]. These methods focus on defense against attack events, and as such mainly involve the stages before and during an attack: modelling the system, the potential attacks and countermeasures; identifying the attack and choosing a countermeasure. We believe that by introducing longer-term resilience considerations, countermeasure investments can be made more efficient. Recent frameworks for attack countermeasure selection have commonly used graphical models [15]. The combined use of attack graphs (AGs) and dependency graphs (DGs) for analysing the impact of cyber attacks has been explored by Albanese et al. [1, 2]. Kotenko and Doynikova [6, 12–14] employ these graph tools in a countermeasure selection system, and Shameli-Sendi et al. [17] use similar graphs for selecting countermeasures in an intrusion response system.

The method in [1, 2] holds the DG alongside the AG as part of the dynamic analysis, with the effects of attack steps on the DG held as key for impact assessment. By contrast, [12] focuses on enriching the system's AG with topological and service dependency information, and using this augmented AG for the analysis. The DG is employed as a source of component importance information at the pre-processing stage. In this work, we follow the approach in [1, 2] of using the DG 'live' during the analysis. This is because for our approach to countermeasure choice and resilience analysis, the dynamic effects from attack steps on the dependencies are required for accurate estimation of costs and loss impacts over time.

Our proposed approach has similarities to the countermeasure selection techniques in [13] and [17] in the use of AGs and DGs, using costs to quantify attack impact, and basing countermeasure decisions on costs. While these works focus on containment in terms of stopping attacks or reducing the risk of them reaching pre-determined goals, we aim to find a strategy that is efficient over longer time, considering the cost of recovery to previous functionality. Additionally, we model time dynamics explicitly, and approach cost and component valuation via the impact on the final services as opposed to giving each component an intrinsic value.

3 IMPACT ANALYSIS MODELLING

3.1 Attack and Dependency Graphs

Attack graphs are a network risk-assessment tool that provide a graphical representation of actions that an attacker can take to reach an attack goal, for example root access on a given server, by exploiting vulnerabilities that exist in the network. Various types of attack graphs have been proposed in the literature, depending on the application. The work presented here will use the definition of Vulnerability Dependency Graph from [1], a compact AG representation used in their work on attack impact assessment.

Definition 3.1. **(Vulnerability Dependency Graph, [1])** *Given a set of vulnerability exploits V, a set of security conditions C, a require relation $R_r \subseteq C \times V$, and an imply relation $R_i \subseteq V \times C$, a vulnerability dependency graph G is the directed graph $G = (V, R)$, where $R = \{(v_i, v_j) \in V \times V \mid \exists c \in C, (v_i, c) \in R_i \wedge (c, v_j) \in R_r\}$ is the edge set.*

The vertices consist of nodes representing vulnerability exploits such as "remote exploit of vulnerability V on host A", while the edges implicitly contain security conditions such as "user access to host A". A require relation R_r between a security condition c and a vulnerability exploit v_i means that c must be satisfied for v_i to be exploited, and an imply relation R_i between v_j and c means that exploit of v_j leads to condition c being satisfied [21]. The edges in the set R link pairs of vulnerabilities that are connected via a security condition by what [2] call a "prepare-for" relation. A prepare-for relation exists between v_i and v_j if v_i has an edge to security condition c representing an imply relation (exploit of v_i implies the attainment of security condition c), and c has an edge to v_j representing a require relation (the exploit of v_j requires the condition c to have been obtained by the attacker). This AG representation leaves the security conditions implicit, as the edges are defined as going through a condition but these are not shown, with the end result resembling a dependency graph of the vulnerabilities.

Dependency graphs (DGs) represent dependency relations between the various components of a system. For example, a server may require input from various databases to perform its function, which themselves may depend on other components. The DGs in this work represent the **availability** dependencies of services provided by software applications in a network. The DG nodes are applications that provide services across the network, such that the functionality of applications elsewhere in the network depends on them. At a given time t, the availability of a given node is measured as the service level it provides at time t as proportion of the level normally expected of it, so that 1 represents full availability, and 0 means the node is unavailable. Dependencies between services are represented by directed edges between the DG nodes. For our DGs, we use the definition of a Generalised Dependency Graph from [1]:

Definition 3.2. **(Generalised Dependency Graph, [1, 2])** *A generalised dependency graph is a labeled directed acyclic graph $D = (H, Q, \phi)$, where: H is a set of nodes, corresponding to network components; $Q = \{(h_1, h_2) \in H \times H | h_1$ depends on $h_2\}$ is a set of edges; $\phi : H \rightarrow \mathcal{F}$ is a mapping that associates with each node $h \in H$ a function $f \in \mathcal{F}$ s.t. the arity of f is equal to the outdegree of h*. For each node $h \in H$, $\dot{h}$ denotes the set of components that depend on h and $\dot{h}$ denotes the set of components h depends on.*

The first two points in the DG definition describe the basic structure of the dependency graph (which nodes depend on which others), while the mapping ϕ describes the type of dependency that each node $h \in H$ has on its supplier nodes $\dot{h}$. We consider the same dependency function types as used by [1, 2]:

$$f_r(a_1, ..., a_n) = \begin{cases} 1 & \text{if } \exists i \in [1, n] \ s.t. \ a_i = 1 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$f_d(a_1, ..., a_n) = \frac{1}{n} \sum_{i=1}^{n} a_i \quad (2)$$

$$f_s(a_1, ..., a_n) = \begin{cases} 1 & \text{if } a_i = 1 \ \forall i \in [1, n] \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

where a_i represents the availability value of a network component which the current component directly depends on, and n is the total number of components on which the current component depends (i.e. for h_i we have $n = |\dot{h}_i|$). Here, f_r is a redundancy-type dependency (logical OR), f_s is strict dependency on all supplier nodes (logical AND), and f_d means availability of h is the mean of the availabilities of its suppliers (degradation). For example, in the sample graph in Fig. 1, for h_T we have $f(s(\dot{h}_T, t)) = f_s(a_1) = f_s(s(h_C, t))$.

3.2 Attack Impact Analysis

Our impact assessment approach builds on the **Impact Assessment Graph** (IAG) proposed in [1], with combined use of an AG and a DG for analysing attack impacts. We propose a slight simplification relative to [1]. Specifically, in the IAG we use an AG without the compromise time-windows feature in the original, as the handling of time in our model makes this feature unnecessary.

Definition 3.3. **(Impact Assessment Graph, modified [1])** *Given a vulnerability dependency graph $A = (V, R)$ and a generalised dependency graph $D = (H, Q, \phi)$, an impact assessment graph is a 4-tuple (A, D, F, η) where: $F \subseteq V \times H$; $\eta : F \rightarrow [0, 1]$ is a function that associates with each pair $(v, h) \in F$ a real number in the $[0, 1]$ interval representing the percentage reduction in the availability of network component h caused by vulnerability exploit v.*

Effectively, the IAG consists of both the AG and the DG in full, and connections in between them in the form of the function η that describes how the availability of the components in the DG are affected by vulnerabilities in the AG.

The **network status function** [1] defines how the status of a node evolves over time with relation to the availabilities of its dependencies. The definition states that a network status function s for a DG D assigns each DG node h a value in the range $[0, 1]$, capped by h's dependency function f over the statuses of its dependencies. That is, the statuses of the components that h depends on define a maximum availability level for h, but the status of h can be below this if directly affected by an attack. However, the functional form of the network node status function $s(h, t)$ in [1] does not allow modelling recovery in a component's status due to the way the previous period's status $s(h, t-1)$ enters the function definition. A low availability status one period will punish the component's status in later periods, even if recovery was done. Therefore, we have redefined the node status function to keep track of all vulnerability exploits that are in effect at time t (either newly exploited, or previously exploited ones whose related security conditions remain compromised), each of which may have a direct impact on h. Our reformulation also introduces a multiplicative interaction between the direct and indirect availability effects, replacing the minimum function in the original. In this way, the direct and indirect effects interact to set the effective availability, in contrast to the dependency availability effect only acting as a cap on the effective availability level. These changes also enable multiple exploits affecting a node $h \in H$ directly, which the original formulation does not support. Our proposed formulation for the node status function is:

$$s(h, t) = f(s(\dot{h}, t)) \prod_{v \in V_{e,t}} (1 - \eta(v, h)) \quad (4)$$

where $V_{e,t}$ is the set of AG nodes that are in an exploited state at time t; $\eta(v, h) \in [0, 1]$ is the availability effect that the exploit of vulnerability v has on component h (0 for no effect, 1 for fully unavailable); $\dot{h}$ is the set of components that h is dependent on; and $f(s(\dot{h}, t))$ is the availability effect on h from its dependencies. Note that a component recovery is represented as the removal of a vulnerability v from the exploited set $V_{e,t}$, while patching vulnerability v removes it the from the full set V. The status of network component h is composed of two effects on availability: 1. Compromise effect (direct): $(1 - \eta(v, h))$, the effect of compromise (vulnerability exploit) in the AG node v which corresponds to component h; 2. Dependency availability effect (indirect): $f(s(\dot{h}, t))$, the effect of unavailabilities in the components that h is dependent on, of type f_r (redundancy), f_d (degradation) or f_s (strict dependence).

We use the example system from [1, 2] to show how our approach differs. The sample represents the network of a small organisation

*"If h is a terminal node in the dependency graph (i.e. it does not depend on any other node), we assume $\phi(h)$ is the constant (0-ary) function 1"[1].

with two final services, an online shopping Web Service (h_A) and a Mobile Order Tracking app (h_C), their local cache databases (h_B and h_D for online shopping and order tracking, respectively), and a separate subnetwork for the internal logic (h_E and h_F) and a central database (h_G) powering the services.

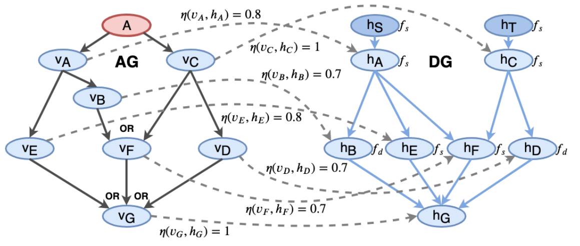


Figure 1: Impact assessment graph for the sample network

The dependency structure and potential attack paths in the sample network from [1, 2] are shown in the IAG in Fig. 1. The AG is on the left, and DG on the right. The AG nodes represent vulnerabilities in the network, with the attacker entry points shown as edges from node A. The DG nodes are network components: h_A to h_G provide intermediate services (internal services), while h_S and h_T are customer-facing services (the product of the organisation). The exploit of a vulnerability v_i affects the availability of the corresponding component h_i, and the availability of components which are dependent on h_i. The dashed lines in Fig. 1 show the availability impact of a vulnerability, $\eta(v_i, h_i)$, for example $\eta(v_F, h_F) = 0.7$ means h_F will lose 70% of its availability when v_F is exploited (unless there is also a dependency availability effect on h_F).

The final key component for attack impact analysis using the IAG is the utility derived from the components in the dependency graph: $\forall h \in H, u(h)$ gives the utility for node h. The utility is, in effect, the value of the service provided by a given node at each time unit. The work in [1, 2] assume that all the DG nodes are given a utility value which remains static during the analysis. By contrast, we believe that intermediate services provide value only when at least a part of the final service to which they contribute is online (available). Therefore, we only set values for the final service nodes (h_S and h_T in the sample), with the value of the other nodes only reflecting the impact they have on the value arising from the final services. In this way, we use the dependency structure to dynamically determine the values of all intermediate services.

In contrast to [1, 2], we propose to introduce a comprehensive cost model that takes into account the costs of node unavailability, costs of different countermeasures, and recovery costs. Doing so, we aim to move beyond the original *attack* impact assessment to a more general impact assessment by including the estimation of the impacts of defensive actions as well as those of the attacker. Countermeasure decisions can then made based on expected costs over time, instead of only relying on the projected impacts of attacker actions. Our work uses the impact analysis framework as part of an approach countermeasure selection, employing the impact estimates for determining which countermeasure is the most cost effective at a given attack situation. For this purpose, leaving out the analysis of costs of the countermeasure actions could lead to inefficient choices of countermeasures, as the application of a countermeasure can reduce the availability of the services.

3.3 Attacks, Countermeasures and Recovery

We model **attacks** as sequences of attack steps, i.e. atomic exploits of a single vulnerability in an AG, with potential to lead to further steps. They enter the network via particular "entry nodes" which are directly exploitable by the attacker. For example, in Fig. 1, an attack can start by an exploit of v_A or v_C. The potential next steps are then determined by the AG structure – for example, if v_A was exploited, the possible next steps are v_B and v_E. The time to exploit a vulnerability v_i (for $i \in V$) is $t_{v_i} = 1$ for all vulnerabilities, so only one attack step can occur during a time unit t. The probability of an attacker taking an attack step at a given time period is p_{step}. Additionally, the parameter $p_{fast-step}$ reflects the chance that a countermeasure/recovery application is slower than the attacker's next step, so the attacker's next step gets executed just before the defender's one. This provides some uncertainty to the timing of events, to avoid limiting us to the case where the defense always beats the attacker to the next step, which seems unrealistic.

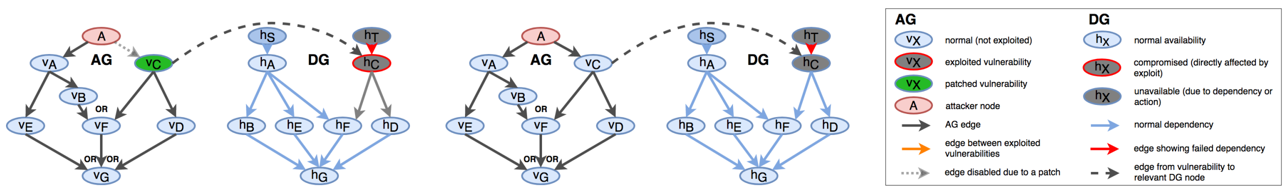
Countermeasures (CMs) are actions aimed at reducing the impact of an attack. In general these could be of two types, ones affecting the network's security capabilities which can be taken before an attack but not during one (capability changes: redundancy additions, back-ups), and others which can be taken at any time (dynamic countermeasures). In this paper, we consider one type of dynamic countermeasure, **patching** vulnerabilities. An another type of dynamic countermeasure, disabling communication links, will be considered in future work, along with capability changes.

The effect of patching is to remove a vulnerability node from the AG, restricting potential attack paths. A patching action requires t_P units of time to implement, and comes at a cost consisting of a direct cost and a service impact, introduced in Sec. 3.4. Fig. 2(a) illustrates the case where v_C is patched after an exploit. Note that patching does not clear the related DG node h_C from compromise, as recovery is handled separately.

Recovery refers to actions that the network owner uses to recover the functionality of components (DG nodes) compromised by vulnerability exploits. To simplify the analysis, we consider the case where there is only one type of recovery method, akin to component replacement. This effectively replaces a compromised component with a clean and working instance, with the same functionality (and vulnerabilities) as before, assumed to take t_R time, and cost c_R. Both the time and cost values are assumed equal across components. In terms of the graphs, this corresponds to making a DG node fully functional and the corresponding AG vulnerability not exploited (although it will remain in the AG, so may be re-exploited). Fig. 2(b) shows the case of recovery of h_C after compromise by exploit of v_C. More detail on recovery modelling is provided in Sec. 3.6.

3.4 Costs of Actions

Our modelling of costs contains direct costs for each action, i.e. node recovery cost c_R and patching cost c_P, in addition to indirect costs in terms of service loss due to dependencies. While the direct costs do matter, the key element of our cost modelling is the loss to the provision of final services caused by component unavailability.


A
AG
DG
v_A
v_B
v_C
v_D
v_E
v_F
v_G
OR
h_S
h_T
h_A
h_B
h_C
h_D
h_E
h_F
h_G
normal (not exploited)
exploited vulnerability
patched vulnerability
attacker node
AG edge
edge between exploited vulnerabilities
edge disabled due to a patch
normal availability
compromised (directly affected by exploit)
unavailable (due to dependency or action)
normal dependency
edge showing failed dependency
edge from vulnerability to relevant DG node

(a) Defender patches v_C (h_C not cleaned automatically). Takes t_P time units to come to effect.

(b) Defender recovers h_C. h_C remains temporarily unavailable. After $t + t_R$, h_C and h_T are up again.

(c) Graph legend

Figure 2: Patching and recovery actions and their effects on the system

Service loss due to unavailability of DG node h at time t:

$$g(h,t) = \sum_{h_j \in H_S} \Big(u(h_j) \cdot \big(f_s(s(\dot{h_j},t) \mid s(h,t) = s(h,t-1)) - f_s(s(\dot{h_j},t) \mid s(h,t)) \big) \Big) \tag{5}$$

where we have used $s(h_j,t) = f_s(s(\dot{h_j},t))$ for $h_j \in H_S$, which follows from (4) and the observation that the customer-facing service nodes h_S and h_T do not have direct exploits (so only the dependency effect counts for them). The function $g(h,t)$ is the impact of h's deviation, at time t, from its previous observed availability level onto the availability of services.

The **costs of the countermeasure and recovery actions** consist of two parts: the direct cost for the action (c_R for recovery of a node, c_P for patching a node, c_D for disabling), and the cost of unavailability of the network components that are directly impacted by the action. For example, the observed (after the fact) cost of patching vulnerability v_i at time t is given by:

$$c_{patch}(v_i,t) = c_P + \sum_{h_j \in M(v_i)} \left(\sum_{\tau=t}^{t+t_P} g(h_j,\tau) \right) \tag{6}$$

where v_i is a node in the AG, c_P is the direct cost of patching. $M(v_i)$ represents the set of elements in the DG that are adjacent to the AG node v_i, that is, the components directly affected by the vulnerability v_i. In other words, these are the software where the vulnerability exists, and where the patching of v_i takes place. The current time period is t, and t_P represents the time units required for the patching. The inner summation adds together the cost of component h_j being unavailable from t to $t + t_P$. The observed costs due to disabling and node recovery work in a similar manner.

Note that (6) shows the calculation of the observed cost when we know the path of any attack steps and defender actions and therefore know $g(h_j,\tau)$ for $\tau = [t, t+t_P]$. For estimating the cost of an action beforehand, we require an expectation of the state of the model in terms of attacker steps and defender actions during the periods in question. In practice, our approach is to estimate the benefit of an action in terms of an expected trajectory for the system state, as explained in Sec. 4.

3.5 Performance Measurement and Resilience

We measure the performance of the networked system with an overall service provision status, **service performance** (SP). It is the weighted sum of the statuses of the client-facing services (the "product" of the organisation), weighted by their relative utilities for the organisation. Mathematically, SP at time t is given as:

$$SP_t = \sum_{h \in H_S} \frac{u(h)}{\sum_{h_k \in H_S} u(h_k)} s(h,t) \tag{7}$$

where $s(h,t)$ is the status (availability level) of component h at time t, $u(h)$ is the utility the organisation derives from the service component h (during its full availability), and H_S is the set of components which are client-facing services. In the sample in Fig. 1 we have $H_S = \{h_S, h_T\}$, as the services are h_S and h_T. When observed over time, this metric can be used to measure the network's resilience based on a performance-curve approach for resilience in the spirit of [3]. From the curve, resilience metrics can be calculated e.g. according to the approaches by [3] or [8].

3.6 Recovery Process

We evaluate our model in a setting with automatic recovery decisions, where the choice of whether to recover a node is based on the likely benefit versus the costs of the recovery strategy. We keep this recovery process separate from countermeasure choices, which simplifies expectation formation for CM selection. Node recovery is done if it leads to a reduction in losses exceeding the recovery cost. The loss reduction from recovering node h at time t is:

$$LR(h,t) = Loss_{\neg R}(h,t) - Loss_R(h,t) \tag{8}$$

where $Loss_R(h,t)$ is the loss with recovery, which is:

$$Loss_R(h,t) = \sum_{\tau=t}^{t_{max}} \mathbb{E}_t(g(h,\tau|s(h,\tau)=0)) + \sum_{\tau=t_{max}}^{t_{horizon}} \mathbb{E}_t(g(h,\tau|s(h,\tau)=1)) \tag{9}$$

where $t_{max} = t + t_R + t_P + t_D$ is the time it would take to recover, patch and re-enable edges (if necessary), and $t_{horizon}$ is the last time period in the time horizon considered. This metric consists of the loss from service unavailability from time t until t_{max} when the recovery is finished, and from time $t_{max} + 1$ onward when the node will be assumed recovered (and not re-compromisable). Loss without recovery, $Loss_{\neg R}(h,t)$, is given by:

$$Loss_{\neg R}(h,t) = \sum_{\tau=t}^{t_{horizon}} \mathbb{E}_t(g(h,\tau|s(h,\tau)=0)) \tag{10}$$

3.7 Sample Impact Analysis for CM Selection

Choosing countermeasures by focusing solely on stopping the progress of an attack means ignoring: 1. the cost of countermeasure actions (direct and indirect), and 2. recovery of the network toward a desired state. Disregard of these aspects can lead to choices that are not efficient in the longer term. Assume that the network in Fig. 1 experiences an attack exploiting v_C at $t = 0$. As v_C is exploited, h_C becomes unavailable, and so does h_T due to its dependency on h_C. While the exploit of v_C already causes considerable damage in terms of service losses, once an attacker has exploited v_C, it can further exploit v_F or v_D. While the exploit of v_D only affects h_D (as h_C is down) resulting in no change in either cost or SP, an attack on h_F takes down the remaining service h_S, causing full loss of service and cost due to service losses of $u(h_T) + u(h_S)$ per time period.

When applied repeatedly at each step, the marginal impact analysis approach from [1, 2] would choose to patch the most high-impact component that could be affected by an attack next. Given the compromise of vulnerability v_C, at the following time step $t = 1$, the next attack step could be v_F or v_D, affecting components h_F and h_D, respectively. Choosing based only on the potential impacts of these attack steps, the choice would be to patch v_F. Given this choice, at time $t = 1$ the attacker could proceed to exploit v_D, which was not patched. The appropriate reaction to this would be to patch v_G, given another round of marginal impact calculation. Finally, to return to full availability of services, nodes v_D and v_C need to be recovered (v_c with patching, v_D without).

While the above approach is sensible if we want to guarantee that the highest impact nodes h_F and h_G are never compromised, it may not be cost efficient over time. The costs of the actions, or recovery, are not considered by the above countermeasure strategy. However, the choice of actions and their order can have a large impact on the costs, especially those arising from service losses. Note that, as the service impact cost $g(h, t)$ for node h is nonzero only when the node causes a change in the status of the services, the service impact of a given node h can change over time. For example, the unavailability of h_G only impacts services when h_F is available. This changing loss impact can make a great difference on the overall cost of a countermeasure strategy. For example, if there is a low probability that the attack will have successfully moved to a different node before the next time step, it may be better to **not** take a countermeasure that contains the attack, but focus on recovering and patching v_C. In the worst case, the attacker has been able to move fast enough to exploit v_F before h_C goes offline for patching and recovery. If so, all services go down at $t = 1$, and h_F will have to be recovered – but this happens without additional availability impact, as services will be down already. Again, in the worst case the attacker may move fast enough to compromise h_G before the system h_F is taken down for recovery, so h_G will require recovery at $t = 3$. Even in this worst case the overall costs could be lower than with the approach from earlier, if the time to recover is lower than to patch $t_R < t_P$. However, this approach also benefits from there being a chance that the attacker will not successfully make another step before the defender reacts, meaning in the best case only h_C has to be recovered and patched.

This sample highlights that, depending on the situation, it can be more cost effective to act reactively and rely on recovery capabilities, while sometimes proactively containing the attack is better. We built our CM selection approach on cost impact analysis to be able to find the approach that works best in a given situation.

4 COUNTERMEASURE SELECTION

Our approach to countermeasure selection relies on cost analysis of the impacts of defender actions and expected attack steps. We shall refer to this method as Cost-Impact Countermeasure Selection, or **CICM**. Our CM selection algorithm essentially estimates the impact of all CMs that apply to an AG node of interest v_s, and returns a list of them in descending order of their overall benefit relative to what would be expected if no countermeasure was applied. The highest benefit CM is implemented, provided that the benefit is positive.

The effectiveness of each countermeasure is evaluated by comparing the expected costs of the CM to the benefits it is expected to yield. Our approach uses the direct cost of the action, and two potential evolutions ("trajectories") of the system: the "expected trajectory" reflects how an attack would be expected to proceed within the network in the absence of the countermeasure, and measures what the impact would be in terms of costs due to both availability loss and recovery actions. The "deviating trajectory" given a countermeasure action measures the expected costs due to availability loss and recovery when the countermeasure is applied. As a countermeasure initially requires a network component to be taken offline temporarily, we make a difference between the immediate and the longer-run impacts of the countermeasure. Therefore, our measure of the benefit arising from a countermeasure is given by:

$$\begin{aligned} B(cm, v_i, t) =& eaf(v_i) \cdot (t_{horizon} - t) \cdot trajD_{LR}(cm, v_i) \\ &+ trajD_{curr}(cm, v_i) - c_{cm} \end{aligned} \tag{11}$$

where the first part is an estimate of benefit in later time steps, consisting of the long-run trajectory difference $trajD_{LR}(cm, v_i)$ multiplied by the expected frequency of future attacks exploiting v_i, $eaf(v_i)$, and the time periods left until the end of the horizon. The second term is the trajectory difference currently (until the CM has been successfully applied), and the last term is the direct cost of the CM, c_{cm}. The expected attack frequency to node v_i, $eaf(v_i)$, is an estimate of the probability that the attacker will attempt to exploit node v_i again. This is not based on the current compromise state, but on the probability to (re-)obtain the privileges for exploiting v_i in the future via any path. We estimate $eaf(v_i)$ by approximating the probability of the shortest viable (not patched) path from the attacker node A to v_i. This we calculate as the step probability p_{step} to the power of the number of edges on the shortest viable attack path from A to v_i. The approximation is by no means fully accurate, but captures the behaviour we want, and is considerably simpler and faster than the calculation of the exact probability. The benefit from the exact calculation would be limited given the randomness included in the model and arising from its environment.

The current and long-run trajectory differences are given by:

$$trajD_{curr}(cm, v_i) = devTraj_{curr}(cm, v_i) - expTraj(v_i) \tag{12}$$

$$trajD_{LR}(cm, v_i) = devTraj_{LR}(cm, v_i) - expTraj(v_i) \tag{13}$$

The expected trajectories $expTraj(v_i)$, $devTraj(cm, v_i)_{curr}$ and $devTraj(cm, v_i)_{lr}$ are calculated by formulating an expected value for availability impact and costs in the next k time steps. For $expTraj(v_i)$, this proceeds as follows: 1. From the AG, estimate

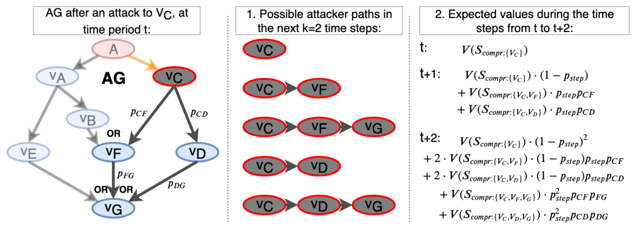

AG after an attack to V_C, at time period t:
A
VA
AG
VC
VB
OR
VE
VF
VD
VG
OR
OR
p_{CF}
p_{CD}
p_{FG}
p_{DG}
1. Possible attacker paths in the next k=2 time steps:
VC
VC VF
VC VF VG
VC VD
VC VD VG
2. Expected values during the time steps from t to t+2:
t: $V(S_{compr:\{V_C\}})$
t+1: $V(S_{compr:\{V_C\}}) \cdot (1 - p_{step})$
$+ V(S_{compr:\{V_C,V_F\}}) \cdot p_{step} p_{CF}$
$+ V(S_{compr:\{V_C,V_D\}}) \cdot p_{step} p_{CD}$
t+2: $V(S_{compr:\{V_C\}}) \cdot (1 - p_{step})^2$
$+ 2 \cdot V(S_{compr:\{V_C,V_F\}}) \cdot (1 - p_{step}) p_{step} p_{CF}$
$+ 2 \cdot V(S_{compr:\{V_C,V_D\}}) \cdot (1 - p_{step}) p_{step} p_{CD}$
$+ V(S_{compr:\{V_C,V_F,V_G\}}) \cdot p^2_{step} p_{CF} p_{FG}$
$+ V(S_{compr:\{V_C,V_D,V_G\}}) \cdot p^2_{step} p_{CD} p_{DG}$

Figure 3: The stages of the expected trajectory calculation

what paths the attacker could follow in the next k time steps starting from node v_i, the current head of the attack. Estimate the impact of each of these potential paths on service availability and on recovery costs. 2. Formulate expected values for the services impact and costs in each of the time steps from t to $t + k$, using the paths calculated in step 1 with the probabilities for each exploit, the probability for attack step being completed in each time step. An illustration of this is provided in Fig. 3, for the case where the vulnerability v_C in the sample graph has been exploited at time period t. The first panel shows the AG situation at time t, the middle panel shows the possible attack paths in the next $k = 2$ time steps. The last panel shows the calculation of the expected values for the time periods $t, t+1, t+2$, where $V(S)$ refers to the valuation (service performance or cost) at system state S. For brevity, the figure uses a shorthand for states with only the exploited nodes listed, e.g. $S_{compr:\{v_C, v_D\}}$ stands for the state where the set of exploited nodes is $\{v_C, v_D\}$. The "deviating trajectory" estimates given a countermeasure cm are calculated similarly to $expTraj(v_i)$, but assuming that a CM is applied. An additional difference is that when calculating the expected values (step 2 above), we further consider the possibility that the CM is not applied in time before the next step by the attacker, represented by the probability parameter $p_{fast-step}$. We calculate two deviating trajectories as the application of a CM causes temporary unavailability at first, before the longer-run effect is obtained.

The split into current and longer term trajectory impacts enables a simultaneous consideration of the immediate up-front costs of a countermeasure (direct cost, temporary availability impact) and the effect it has on the robustness of the network in future periods. Our method accounts for the future by calculating the "expected trajectory" impact and the expected number of times such an attack would be observed in the future. This approach provides an estimate of longer term impact that is considerably more efficient to calculate than directly considering all possible states in each future period. A weakness of the method is that it considers future impacts relative to the current situation, so excludes the impact of changes to the network or the environment that might happen in the future. However, such information is unlikely to be available at the time when the decision is to be made. In the case it was, it should be possible to establish the robustness of the estimate to such changes, e.g. by calculating the impact of the CM conditional on other CMs being applied as well. However, for the sake of tractability and scalability, we followed a greedy strategy looking for the impact of the countermeasures independently.

While we calculate probabilities of various paths of attacks to formulate the expected trajectory, we do not solve for the whole network or the full time horizon, as such an approach would not scale to realistic network sizes. Instead, the method focuses on "neighborhoods" of the current attack, by looking at potential states a few time-steps forward from the current "boundary" of the attack. As another way to simplify the problem, we don't consider patching at each of the nodes, but focus on the nodes nearest to the entry (if patchable) [for future attacks], and those on the attack boundary [for containing the ongoing attack, as well as future attack steps].

5 EVALUATION

We used simulations to investigate the usefulness of the proposed framework for countermeasure selection, by comparison to two alternative strategies. The first comparison point is what we call the Attack Impact Approach (**AIA**), which we built by adapting the attack impact assessment approach from [1] to use in automated countermeasure selection. While their original work was not proposed for this specific purpose, we built a CM selection approach for patching actions using the main principles of their attack impact assessment method, using their marginal impact metric for patching choices. To use this in a setting where we care about performance over time, we also required the method to handle recovery, so included our version of the node status calculation in AIA. AIA can be considered a containment approach, as it chooses patching actions that apply to the vulnerabilities exploitable next by the attacker, not ones that apply to already exploited vulnerabilities. This has the potential for stopping the attacker from compromising important nodes, but applying CMs on healthy components will lead to temporary availability losses, which can be costly.

The second comparison is to a strategy where a patch is always applied to the latest exploited vulnerability, without considering costs or alternative actions. We call this **PLE**, for "Patch Latest Exploit". This approach rejects containment in favour of blocking the last used attack paths from being exploitable in the future, effectively limiting future exposure while accepting current risk. PLE benefits from limiting the availability loss from CM actions, as actions are only applied to nodes that already suffer from reduced availability. While the two comparison points represent extremes in terms of containment and treatment focus, our approach is intended to be able to choose a cost-efficient approach in between these extremes based on the attack situation. The same recovery process, described in section 4, is used with all of the approaches.

We ran tests under two settings: the sample graph in Fig. 1, and randomly generated synthetic graphs. The sample graph provides a useful basis for comparisons as the attack impact assessment method by [1, 2] was demonstrated on it. The generated graphs are directed acyclic graphs of a specified size (in terms of nodes). For simplicity, the number of AG nodes was restricted to match that of the DG nodes. To control the structure, we have restricted the maximum number of parents of a node (nodes dependent on the node) to three. The connections between the AG and DG nodes are chosen at random, meaning that the vulnerabilities in the AG correspond to random system components (in DG), and the attack paths on the AG can be considerably different from paths in the dependency structure. Furthermore, in the DG, the number of dependencies (children) of each node are chosen at random, as are the dependency functions for each node. For each DG, two nodes are allocated as service nodes. In the AG, the number of children of a given node (number of further vulnerability exploits made possible by a given exploit) is drawn at random. To introduce attack entry points, we add a node representing the attacker's starting point, and its children drawn among the other nodes are the entry nodes. Additionally, probability values for the AG edges are set, representing the ease of exploiting a node, drawn from a distribution corresponding to the access complexity metric of CVSS scores [7].

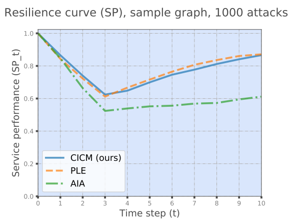
We simulated randomly generated attacks into each graph in question. The attacks follow a path towards a goal node, which is picked from among the AG nodes with the highest availability impact on the final services (the single highest impact one, or drawn among the shared highest impact nodes). Each chosen edge is picked from those along the paths to the goal, based on a draw between viable candidates, where the distribution is based on the edge probability values (representing access complexity). This creates variety across simulations, approximating different attacker choices based on e.g. different skill levels. One vulnerability exploit is allowed per time period. If a given step is not possible, due to a CM action, the attacker attempts another exploit that is on a path to the goal. If the goal becomes unreachable, the attack will stop.

Unless otherwise stated, the simulations use the following parameter values: one unit of time to compromise a vulnerability $t_{v_i} = 1$, two units to patch a vulnerability ($t_P = 2$), and one to recover a node $t_R = 1$. The direct costs are $c_P = 2$ for patching, and $c_R = 3$ for node recovery. There is a 30% probability that an attacker takes an attack step at a given time step ($p_{step} = 0.3$), and $p_{fast-step} = 0.3$, so there is a 30% chance that a given CM/recovery application is slower than the attacker's next step.

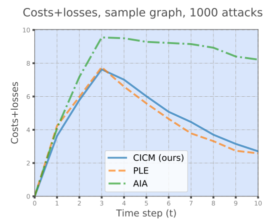
5.1 Results for the Sample Graph

The simulation results on the sample graph are shown in Fig. 4. The figure shows a comparison of our method to the two alternative CM schemes, both for the resilience curve (SP metric), and for costs and service losses over time. The curves represent the mean values for the metrics (SP, overall costs) across 1000 simulated attacks; the solid curve indicates our method (CICM), the dashed one is for PLE and the dash-dot line for AIA. The two upper panels show simulations where the attacks are detected immediately at the first step, and the defender actions (countermeasures, recovery) can be started immediately. By comparison, on the lower two panels, there is a delayed detection of the initial attack step, in which the attacker has already done one step before a step is detected (that is, the second step overall) and the defender actions start. We believe that this case is closer to reality, as attacks can remain undetected in a network for long periods of time.

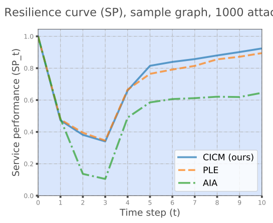
Comparing the results for our approach to those from AIA, we notice our method outperforming AIA both with regard to service performance over time, and the overall costs. The difference between our method and AIA becomes large early on and fails to recover afterwards. This is true whether we look at the case with immediate attack detection (upper panels), or the case with delayed detection (lower panels). Using our approach, the mean SP was over 20% better (23% for immediate detection, 27% for delayed), and the overall costs were 42% lower on average than for AIA (for both immediate and delayed detection). The Wilcoxon signed-rank



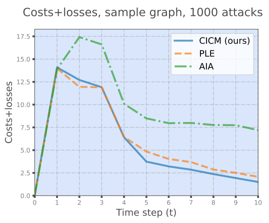
(a) Resilience curve (SP); fully detected steps



(b) Costs and service losses over time; fully detected steps



(c) Resilience curve (SP); 1 undetected step



(d) Costs and service losses over time; 1 undetected step

Figure 4: Simulation results for the sample graph

test rejected the hypothesis of the difference being zero for SP and overall costs in both immediate and delayed case, suggesting that the performance difference is statistically significant. The result appears to be due to reduced service availability, which causes the similarity in the patterns of SP and overall costs (of which service losses is a part). As these are average curves over randomised attacks, some of the detail related to the individual runs is averaged out, but it seems that AIA has a harder time purging the attack from the system than the other approaches, so the difference to other approaches remains until the end of the time window shown.

The PLE approach exhibits performance roughly matching ours, both when the detection of the attack is instant (upper panels), and when there is a delay (lower panels). In fact, the difference between the two approaches was found statistically insignificant using the Wilcoxon test. For this particular graph, this result can be expected. As discussed in Sec. 3.7, to obtain good CM selection performance, the initial exploit will always be patched in this sample graph as the services are directly and fully dependent on the entry nodes. Additionally, the small size of the sample graph limits the variety in attacker steps, and thus on the CM choices of the defender.

5.2 Results for Randomly Generated Graphs

Simulations were run on graphs of different sizes, and varying the detection delay between no delay and a delay of two steps. The size classes vary the number of nodes in the DG (10, 20 and 50-node DGs), with a corresponding number of AG nodes. We generated 100 graphs of each size, and ran 100 random attacks on each graph.

Table 1 shows the results for comparisons to the AIA approach in the case of immediate detection. The results for delayed detection are overall very similar, so we have omitted them to save space. We found a considerable benefit for CICM in terms of both the SP metric and overall cost. While the size of the difference in overall costs

Table 1: Comparing mean values of performance metrics, our approach (CICM) vs AIA; immediate detection

	CICM			Difference: CICM - AIA					
DG size	10	20	50	10		20		50	
				diff.	# +/-	diff.	# +/-	diff.	# +/-
SP	0.862 (0.301)	0.904 (0.243)	0.946 (0.169)	0.168 (0.352)	95/3	0.132 (0.298)	95/2	0.077 (0.224)	100/0
p-value	-	-	-	**0.00**	-	**0.00**	-	**0.00**	-
Cost	3.49 (7.14)	2.71 (5.92)	1.84 (4.32)	-3.91 (8.15)	4/96	-3.25 (7.00)	0/100	-2.17 (5.33)	0/100
p-value	-	-	-	**0.00**	-	**0.00**	-	**0.00**	-

Notes: 100 graphs per size, 100 attack simulations each; "# +/-": count of positive/negative diffs.

Table 2: Comparing mean values of performance metrics, our approach (CICM) vs PLE

Immediate attack detection

	CICM			Difference: CICM - PLE					
DG size	10	20	50	10		20		50	
				diff.	# +/-	diff.	# +/-	diff.	# +/-
SP	0.862 (0.301)	0.904 (0.243)	0.946 (0.169)	0.001 (0.112)	42/19	0.000 (0.110)	38/30	0.000 (0.080)	29/38
p-value	-	-	-	**0.01**	-	0.17	-	**0.00**	-
Cost	3.49 (7.14)	2.71 (5.92)	1.835 (4.32)	-0.03 (2.69)	34/58	-0.03 (2.59)	43/56	0.04 (2.23)	36/64
p-value	-	-	-	**0.00**	-	**0.00**	-	**0.02**	-

Delayed attack detection, two undetected steps

	CICM			Difference: CICM - PLE					
DG size	10	20	50	10		20		50	
				diff.	# +/-	diff.	# +/-	diff.	# +/-
SP	0.801 (0.361)	0.852 (0.302)	0.901 (0.235)	0.044 (0.235)	78/19	0.034 (0.197)	71/21	0.020 (0.154)	68/25
p-value	-	-	-	**0.00**	-	**0.00**	-	**0.00**	-
Cost	5.04 (8.86)	4.16 (7.60)	3.33 (6.32)	-1.09 (5.60)	11/86	-0.96 (4.85)	8/91	-0.80 (4.08)	6/94
p-value	-	-	-	**0.00**	-	**0.00**	-	**0.00**	-

Notes: 100 graphs per size, 100 attack simulations each; "# +/-": count of positive/negative differences.

between the approaches gets smaller with the size of the graphs, due to a general improvement of both of the approaches, the relative cost saving for CICM is around 53-54% across graph sizes for immediate, and 46-51% for delayed detection. Using the Wilcoxon signed-rank test, we find that CICM demonstrated statistically significantly better results than AIA for both of the metrics.

Comparisons to the PLE strategy are displayed in Table 2. When there is no delay in attack detection, CICM performs, on average, almost identically to PLE in terms of the performance metric SP. The difference values are practically zero, and the Wilcoxon test fails to reject the null hypothesis of equal means for SP for the 20-node case. On the side of overall cost, the difference magnitudes are also small. While the simulations show CICM slightly outperforming PLE on average for 10 and 20-node graphs, with 50-node graphs the results are mixed. Overall, CICM was more cost-efficient in 64% of the 50-node graphs simulated, but the mean difference over all graphs is in favour of PLE (2% cost difference). The main message here is: if the attack is detected immediately at the point of entry, there is little difference between CICM and PLE. This makes sense, as patching the first node stops the attacker from regaining access to the network after the attack is purged elsewhere, and immediate detection makes this very effective.

When there is a delay in the detection of the attack, the benefits of CICM over PLE become clear. There is some evidence of a benefit on the average service performance relative to PLE (amounting to between 0.2% and 0.5% of the overall value of the output of the services, on average). Regardless, the main impact is on the overall cost side, where our method provides average efficiency improvements over PLE amounting to 18% for 10-node, 19% for 20-node and 20% for 50-node graphs. Importantly, the cost savings are consistent, with benefits obtained in 90% of all the graphs tested (see the +/- counts in the lower part of Table 2: 86% of 10-node graphs had a negative sign for the cost difference, 91% of 20-node graphs, and 94% of 50-node). The fact that the difference shows up in cost instead of SP makes sense, as our method chooses actions based on the overall cost, not on SP. The results also show that this effect grows with the size of the graph, suggesting that larger graphs provide more room for choices that improve cost efficiency.

Summarising the findings from Table 2, we conclude that while a straightforward patching strategy like PLE can yield good results in terms of service performance, our method provides considerable cost savings when attacks are detected with a delay.

We investigated the sensitivity of the results to the length of delay in detection. We varied the number of attack steps that go undetected before the defender starts their CM selection and recovery processes, with the other parameter values held constant at the levels mentioned above. For this, we used the 20-node generated graphs (100 graphs, 100 simulated attacks for each graph), comparing to PLE. The results show that the cost savings from CICM relative to PLE increase drastically when moving from immediate to delayed detection, with the saving jumping from 1.1% for immediate detection (0 undetected steps) to 11% for one and 19% for two undetected steps. However, further undetected steps provide no additional advantage to our approach, with the relative cost decreasing slightly to 18% and 17% for 3 and 4 undetected steps, respectively. The reason for the initial jump is clear, as the attacker holds more ground in the network and has more possibilities to pursue, and the PLE strategy loses its edge when there are choices to be made. The reduction to the benefit at higher number of undetected steps is due to an increase in average overall cost faced by both approaches, so the difference is smaller relative to this level.

Sensitivity of the results to different assumptions about the cost structure was also tested. These were again run for the 20-node graphs, and for delayed detection with 2 undetected steps, which provided the highest cost impact in the delay sensitivity tests. The tests consisted of varying two different settings relating to costs: the utility obtained from services per time unit, which affects the indirect costs arising from node unavailability; the ratio of the direct cost of recovery to the direct cost of patching, which can affect the cost-effectiveness of patching relative to recovery actions.

Fig. 5 shows the results for the cost parameter sensitivity. We can see two broad patterns: First, the magnitude of the cost difference between the approaches is reduced as the average service utility level increases. Second, in most cases a higher cost of recovery (higher c_R/c_P ratio) leads to an increase in the cost savings provided by CICM compared to PLE, other things equal.

The first observation suggests that the higher the potential loss from unavailability, the less room there is to find benefit from actions that proactively contains the attack as opposed to treating the latest compromise. Therefore, the cost-efficient approach becomes more similar to PLE, which not only treats the latest event but also

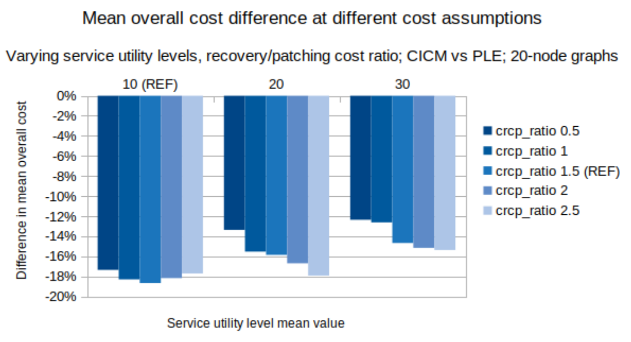


Figure 5: Sensitivity to different cost parameter levels

patches only nodes that are already down, avoiding additional unavailability costs. However, the difference between the approaches is still sizeable, with the smallest difference in Fig. 5 suggesting a 12% average saving using CICM relative to PLE.

The second observation suggests that the higher the cost of recovery is relative to patching, the more room there is for CICM to find cost savings by deviating from treating the latest compromise, as the cost of taking proactive patching steps is reduced. The average cost levels of both approaches increases with the c_R/c_P ratio, but the rise in the costs incurred is smaller when using CICM than PLE, leading to an increasing relative benefit.

6 CONCLUSION AND FUTURE WORK

We proposed a framework for automated countermeasure selection based on cost impact analysis of the organisation's service loss and costs over a period of time considering both attacker and defensive actions, aiming for a cost-effective approach to maintaining service functionality. The method was demonstrated via examples in a sample network, and an evaluation of its countermeasure selection performance was conducted using simulations. The results suggest that our method outperforms an alternative countermeasure selection approach based on the attack impact assessment method by [1, 2], both in terms of average service performance and overall costs over a given time window. Comparisons against a straightforward patching approach showed that, while average service performance was a close match, our method found more cost-efficient ways to achieve the goal.

Future work will involve adding more detail into the impact modelling, moving away from the simple DG as used here in favour of a more flexible dependency model. On the countermeasure selection side, we shall investigate extending the method to select a combination of countermeasures and recovery actions in the same decision. We also intend to extend the method to also consider detailed disabling schemes based on network connectivity. The current model could estimate the impact of such schemes, but we do not currently have this additional topological information included in the attack or dependency graphs. Consequently, a way to model and evaluate the system's performance with such disabling schemes is required.

REFERENCES

[1] M. Albanese and S. Jajodia. 2017. A Graphical Model to Assess the Impact of Multi-Step Attacks. *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* 15, 1 (2017), 1–15.

[2] M. Albanese, S. Jajodia, A. Pugliese, and V. S. Subrahmanian. 2011. Scalable Analysis of Attack Scenarios. *Atluri V., Diaz C. (eds) Computer Security – ESORICS 2011. Lecture Notes in Computer Science* 6879 (2011), 416–433.

[3] M. Bruneau, S.E. Chang, R.T. Eguchi, G.C. Lee, T.D. O'Rourke, A.M. Reinhorn, M. Shinozuka, K. Tierney, W.A. Wallace, and D. Von Winterfeldt. 2003. A framework to quantitatively assess and enhance the seismic resilience of communities. *Earthquake spectra* 19, 4 (2003), 733–752.

[4] S. Choudhury, L. Rodriguez, D. Curtis, K. Oler, P. Nordquist, P. Chen, and I. Ray. 2015. Action Recommendation for Cyber Resilience. In *Proc. 2015 Workshop on Automated Decision Making for Active Cyber Defense (SafeConfig '15)*. ACM, 3–8.

[5] M. Chronopoulos, E. Panaousis, and J. Grossklags. 2018. An options approach to cybersecurity investment. *IEEE Access* 6 (2018), 12175–12186.

[6] E. Doynikova and I. Kotenko. 2015. Countermeasure Selection Based on the Attack and Service Dependency Graphs for Security Incident Management. *LNCS, Int. Conf. Risks and Security of Internet and Systems CRiSIS 2015* 9572 (2015), 107–124.

[7] FIRST. 2018. Common Vulnerability Scoring System. (2018). https://www.first.org/cvss, Accessed: 2018-09-14.

[8] A.A. Ganin, E. Massaro, A. Gutfraind, N. Steen, J.M. Keisler, A. Kott, R. Mangoubi, and I. Linkov. 2016. Operational resilience: Concepts, design and analysis. *Scientific Reports* 6 (2016), 1–12.

[9] L.A. Gordon and M.P. Loeb. 2002. The economics of information security investment. *ACM Trans. Inf. Syst. Security (TISSEC)* 5, 4 (2002), 438–457.

[10] L.A. Gordon, M.P. Loeb, and W. Lucyshyn. 2003. Information security expenditures and real options: A wait-and-see approach. *Computer Security Journal* XIX, 2 (2003), 1–16.

[11] S. Hosseini, K. Barker, and J.E. Ramirez-Marquez. 2016. A review of definitions and measures of system resilience. *Reliab. Eng. Syst. Saf.* 145 (2016), 47–61.

[12] I. Kotenko and E. Doynikova. 2014. Evaluation of computer network security based on attack graphs and security event processing. *J. Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* 5, 3 (2014), 14–29.

[13] I. Kotenko and E. Doynikova. 2015. Countermeasure selection in SIEM systems based on the integrated complex of security metrics. In *Proc. 23rd Euromicro Int. Conf. on Parallel, Distrib., and Network-Based Processing, PDP 2015*. IEEE, 567–574.

[14] I. Kotenko and E. Doynikova. 2016. Dynamical Calculation of Security Metrics for Countermeasure Selection in Computer Networks. In *Proc. 24th Euromicro Int. Conf. on Parallel, Distrib., and Network-Based Processing, PDP 2016*. IEEE, 558–565.

[15] P. Nespoli, D. Papamartzivanos, F.G. Marmol, and G. Kambourakis. 2017. Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks. *IEEE Commun. Surveys Tut.* 20, 2 (2017), 1–37.

[16] L.L. Njilla, C.A. Kamhoua, K.A. Kwiat, P. Hurley, and N. Pissinou. 2017. Cyber Security Resource Allocation: A Markov Decision Process Approach. In *High Assurance Systems Engineering (HASE), 2017*. IEEE, 49–52.

[17] A. Shameli-Sendi, H. Louafi, W. He, and M. Cheriet. 2016. Dynamic optimal countermeasure selection for intrusion response system. *IEEE Trans. Depend. Sec. Comput.* 15, 5 (2016), 755–770.

[18] T.A. Shatto and E.K. Cetinkaya. 2017. Variations in Graph Energy: A Measure for Network Resilience. In *Resilient Networks Design and Modeling (RNDM), 2017 9th Int. Workshop on*. IEEE, 1–7.

[19] J.P.G. Sterbenz, E.K. Çetinkaya, M.A. Hameed, A. Jabbar, S. Qian, and J.P. Rohrer. 2013. Evaluation of network resilience, survivability, and disruption tolerance: analysis, topology generation, simulation, and experimentation. *Telecommunication systems* 52, 2 (2013), 705–736.

[20] J.P.G. Sterbenz, D. Hutchison, E.K. Çetinkaya, A. Jabbar, J.P. Rohrer, M. Schöller, and P. Smith. 2010. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks* 54, 8 (2010), 1245–1265.

[21] L. Wang, S. Noel, and S. Jajodia. 2006. Minimum-cost network hardening using attack graphs. *Computer Communications* 29, 18 (2006), 3812–3824.

[22] N. Yodo, P. Wang, and M. Rafi. 2018. Enabling Resilience of Complex Engineered Systems Using Control Theory. *IEEE Trans. Rel.* 67, 1 (2018), 53–65.