

Resource-aware Cyber Deception in Cloud-Native Environments

Marco Zambianco, Claudio Facchinetti, Roberto Doriguzzi-Corin, Domenico Siracusa
Cybersecurity Centre, Fondazione Bruno Kessler, Trento - Italy

Abstract—Cyber deception can be a valuable addition to traditional cyber defense mechanisms, especially for modern cloud-native environments with a fading security perimeter. However, pre-built decoys used in classical computer networks are not effective in detecting and mitigating malicious actors due to their inability to blend with the variety of applications in such environments. On the other hand, decoys cloning the deployed microservices of an application can offer a high-fidelity deception mechanism to intercept ongoing attacks within production environments. However, to fully benefit from this approach, it is essential to use a limited amount of decoy resources and devise a suitable cloning strategy to minimize the impact on legitimate services performance. Following this observation, we formulate a non-linear integer optimization problem that maximizes the number of attack paths intercepted by the allocated decoys within a fixed resource budget. Attack paths represent the attacker’s movements within the infrastructure as a sequence of violated microservices. We also design a heuristic decoy placement algorithm to approximate the optimal solution and overcome the computational complexity of the proposed formulation. We evaluate the performance of the optimal and heuristic solutions against other schemes that use local vulnerability metrics to select which microservices to clone as decoys. Our results show that the proposed allocation strategy achieves a higher number of intercepted attack paths compared to these schemes while requiring approximately the same number of decoys.

I. INTRODUCTION

Cloud-native applications running on highly-distributed environments with a variety of interfaces and microservices inevitably increase the attack surface of the infrastructure [1]. Traditional cyber defense mechanisms, like intrusion detection systems (IDSs), rely on up-to-date knowledge of attacker tactics, techniques, and procedures (TTPs), which frequently evolve due to the multitude of technologies employed [2]. As a result, production applications are more exposed to zero-day attacks where deployed microservices can be exploited by malicious actors as a foothold to further spread within the target environment.

To strengthen security, cyber deception can be used as a proactive defense strategy by allocating decoys that resemble legitimate system components within the defender infrastructure to lure malicious actors into interacting with them [3]. Decoys must appear realistic to engage the attacker and increase the likelihood of interaction. Traditionally, decoys are general-purpose (possibly vulnerable) applications that are created a-priori and shipped to customers and, as such, they might have little or no fit within the defender ecosystem [4]. Moreover, when prompted by an attacker they cannot ensure the same degree of similarity or interactivity level as the original applications [5] [6] [7]. On the other hand, container-

ization techniques and cloud-native technologies provide the instruments to create high-fidelity and high-interaction decoys by cloning microservices of deployed applications with limited complexity [8]. Unlike pre-built decoys, which require a non-negligible management overhead and are usually installed as standalone elements within the defender environment [9], deceptive microservices can be seamlessly deployed and monitored in production using cloud orchestration technologies like Kubernetes.

By blending this deception mechanism with the running microservices, defenders can detect and mitigate internal threats such as attacker lateral movements between microservices while gathering valuable information about the attacker TTPs. Moreover, using clones of organization’s microservices as decoys reduces the chance that attackers will unveil the deception mechanism since they are more likely to believe that the decoys are legitimate system components. This belief is further enforced by the fact that replication of microservices is a well-known horizontal scaling technique adopted to accommodate variable workload conditions [10]. Indeed, the attacker might expect to find multiple copies of the same microservice within the system configuration.

To fully benefit from this deception approach, two main challenges must be addressed. Firstly, enterprises can’t afford a massive deployment of decoys within their infrastructure, due to limited computational resources, potential scarcity (e.g., in edge scenarios), and associated costs (capital and/or operational expenditure). A key requirement for a practical deception solution is therefore balancing the number of decoys based on incurred resource usage. Secondly, unlike traditional decoy placement schemes that consider decoys isolated from the target environment, a deceptive ecosystem based on copies of existing microservices is intertwined with the real application deployment. Consequently, the decoy allocation strategy (i.e., which and how many production microservices are cloned) plays a fundamental role in taking advantage of this behavior and maximizing the chance of attacker interaction. An effective decoy placement scheme should consider the modification introduced by decoys on the original microservice deployment, tailoring the cloning mechanism to the altered perception of the attacker about the target environment.

Although research on decoy placement schemes has addressed some of the challenges related to this allocation problem, it has limitations. Most of the proposed strategies assume that decoys can be located near the organization’s network perimeter and thus cannot handle threats that elude the first line of defences or insider attacks propagating within the production environment [4]. Additionally, these schemes often

consider a fixed number of decoys that need to be allocated on a predetermined number of system assets. As a consequence, the scaling efficiency as well as the resource consumption of these techniques is not addressed and cannot be extended to support large-scale microservice deployments where every microservice can be potentially cloned as decoy. Finally, many of these approaches rely on game-theoretical models to determine a suitable decoy allocation strategy [11] [12]. Such resolution methodology cannot properly capture neither the heterogeneous nature of the resources (e.g. CPU, memory, disk storage) nor the constraint on the maximum availability of those resources in the considered physical infrastructure.

Motivated by such research gap, this paper presents a novel optimization-based decoy allocation scheme for cloud-native microservice architectures. In detail, our work provides the following contributions:

- we design a metric to evaluate the decoy effectiveness in luring attacks according to the attack graph structure. The latter models the admissible lateral movements of an attacker between microservices. The proposed formulation accounts for the topology modifications in the attack graph introduced by the decoy deployment in order to fully characterize the deceptive capability of each new decoy placement.
- we formulate an integer non-linear optimization problem leveraging the proposed metric in order to compute the optimal decoy placement strategy. By including the impact of the allocated decoys on the attack graph within the objective function formulation, our solution maximizes the number of attack paths intercepted by the decoys according to the availability of computational resources.
- we design an heuristic scheme to overcome the computational complexity of the optimal formulation. The proposed algorithm approximates the original objective function by leveraging an iterative decoy placement strategy which greatly reduces complexity while ensuring a modest performance loss compared to the optimal solution.
- we evaluate optimal and heuristic solutions against different schemes using several metrics to assess the effectiveness of our resource-aware decoy placement strategy. Our schemes outperformed the benchmark ones in terms of deceptive capability performance, while requiring approximately the same number of decoys.

The remainder of the paper is structured as follows. In Section II, we discuss the related work. In Section III, we describe the system model and each component. In Section IV, we propose an optimization-based decoy allocation scheme as well as a heuristic algorithm to approximate the optimal solution. In Section V, we assess the performance of the proposed schemes. Finally, we draw the conclusion in Section VI.

II. RELATED WORK

Modern cyber deception has evolved towards the design of flexible and adaptive deception techniques tailored to the considered defense scenario in order to maximize the deception effectiveness and reduce the management cost complexity. A

more detailed discussion of cyber deception principles and related challenges can be found in [13] and [14]. In the context of microservice architectures, deception strategies need lightweight and flexible decoy implementations to minimize the impact on the computational resources [15]. Moreover, the distributed nature of microservice-based applications requires the design of fine-granularity decoy allocation schemes to efficiently cover the wide attack surface characterizing this technology [16]. Following these observations, we discuss the work on decoy/honeypot allocation schemes by focusing the analysis on the related resource efficiency and deceptive effectiveness of the decoy placement.

The authors of [17] develop a sandbox network that misdirects attacks from the production network towards deceptive containers with exposed vulnerabilities that are deployed in a isolated and monitored virtual environment. Although their approach allows to trap the attacker on a confined environment separated from the real assets, both the management cost and the decoy resource consumption performance are not addressed. We instead limit the orchestration complexity by directly disseminating deceptive microservices replicas within the production infrastructure according to a given resource budget.

In [18], the authors propose a fingerprint anonymity technique to slow and increase the difficulty of the attacker reconnaissance activities by generating container replicas having a fake set of labels attributes. Differently, our decoy allocation scheme deceives the attacker lateral movements between microservices by employing fully-interactive microservice replicas that are clone of the legitimate ones.

The authors of [19] and [20] design a deep reinforcement learning agents that place fake microservices replicas in order to maximize their capability to lure attacks directed toward legitimate microservices and to conceals assets, respectively. Similarly, [21] proposes a two-phase honeypot allocation algorithm combining game theory and reinforcement learning techniques in order to model and dynamically adapt the honeypot allocation according to the attacker activity. Beside the omission of the decoy resource consumption, the main limitation of [19] [20] [21] is the scaling efficiency of the proposed schemes that mostly derived from machine learning approaches. Specifically, these algorithms suffers from a high computational complexity as the training procedure requires many iterations to converge toward a stable allocation policy when the number of possible decoy allocation configurations increases. Instead, we tackle the design of a resource-aware decoy allocation scheme using an optimization-based approach that allows to extend the solution for a high number active microservices by leveraging a low-complexity heuristic.

The authors of [22] design a honey-credential deployment strategy that allocates fake account/password to maximize the probability to lure attacks. Although their approach ensures a very low resource usage as the decoy are simple text files, our solution still ensures a manageable resource consumption overhead and, at the same time, high-interaction decoys that can provide valuable information on the attacker TTPs when violated.

In [23], the authors formulate a honeypot placement scheme

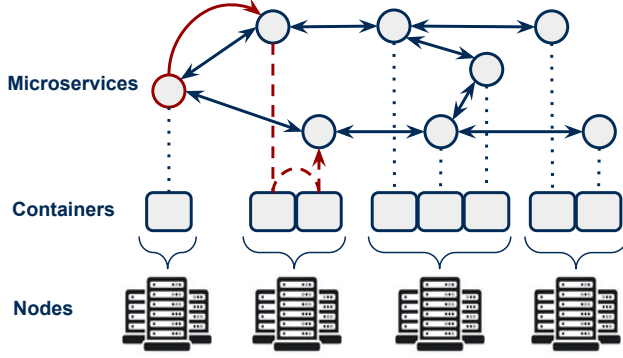


Fig. 1: Overview of the considered system model. Microservices can be violated by means of remote code execution techniques (solid red arrow) or by container escape techniques (dashed red arrow).

given a limited budget of resources. In particular, the resulting allocation policy leverages a game-theoretical approach and incentivizes the placement of decoys on the most valuable network assets. However, the considered resources are homogeneous whereas we consider heterogeneous computational resource such CPU cycles and RAM volume needed to run each decoy. Furthermore, the placement strategy follows a sidecar approach which aims at defending vulnerable network entities by exclusively allocating decoys on those assets. Differently, we aggregate the exploit difficulty of the various microservices in order to place decoys on critical microservices whose violation would allow the attacker to spread more effectively among the infrastructure.

Finally, the authors of [24] and [25] propose statistic approaches based on probabilistic attack graphs and partially observable Markov decision process, respectively, in order to provide a decoy allocation strategy according to the attacker uncertain behavior. However, they assume that the decoy placement is restricted to specific locations within the infrastructure and neglect the impact of the decoys on the attack graph topology, thus limiting the approaches flexibility. Differently, we assume that any microservice can be replicated as decoy and our proposed metric also models the topology modifications on the attack graph due to the decoy placement based on the worst-case attack path scenario.

III. SYSTEM MODEL

We discuss the system model that we considered in order to design the decoy allocation strategy. We provide a comprehensive representation of the main system features in Fig.1. In the following subsections, we describe the various components identified as threat model, microservice architecture model, decoy configuration and placement model and attacker model. In Table I, we report the model notation.

A. Threat model

We consider an attack scenario where a single “pivot” adversary has compromised a microservice of an application hosted by some organization. The attacker exploits the vulnerabilities

TABLE I: Glossary of symbols.

Sets	
N	Set of physical nodes
M	Set of microservices deployed in a given infrastructure
D	Set of allocated decoys
V	$V = M \cup D$ set of vertices in the attack graph
E	Set of directed edges in the attack graph
Parameters	
$d_m^{(j)}$	$d_m^{(j)} \in D$ is the j -th decoy replica of microservice $m \in M$
v_i	$v_i \in V$ vertex in the attack graph
e_{ij}	$e_{ij} = (v_i, v_j)$ edge between vertices v_i and v_j in the attack graph
C_n^{cpu}	Nominal CPU resources of node $n \in N$
C_n^{ram}	Nominal RAM memory available on node $n \in N$
$r_{m,n}^{cpu}$	CPU cycles requested by microservice $m \in M$ on node $n \in N$
$r_{m,n}^{ram}$	RAM requested by microservice $m \in M$ on node $n \in N$
$\Delta C_n^{(\cdot)}$	Amount of available resources (CPU or RAM) on node $n \in N$
δ	$\delta \in [0, 1]$ percentage of available resources reserved to deception
EM_ν	Degree of exploitability of vulnerability ν
ECM_ν	Degree of likelihood of vulnerability ν being attacked
$AP_{(v_s, v_t)}$	Attack path between vertices v_s and v_t
$DAP_{(v_s, v_t)}$	Deceptive attack path between vertices v_s and v_t
$\sigma_{AP_{(v_s, v_t)}}^{(v_i)}$	Binary function such that $\sigma_{AP_{(v_s, v_t)}}^{(v_i)} = 1$ iff $v_i \in AP_{(v_s, v_t)}$
$\bar{\sigma}_{AP_{(v_s, v_t)}}^{(v_i)}$	Binary function such that $\bar{\sigma}_{AP_{(v_s, v_t)}}^{(v_i)} = \sigma_{AP_{(v_s, v_t)}}^{(v_i)}$ iff $D = \emptyset$
$\bar{b}(v_i)$	Degree of betweenness of vertex $v_i \in V$
Decision variables	
x_m	Number of decoys cloning microservice $m \in M$

of such microservice as entry-point to further penetrate the microservice infrastructure and steal important data. The attacker initial access is unknown to the defender (i.e. the organization) and can be provided by any hardware or software element connected to the infrastructure like the organization website, servers, phishing mails, employee laptops and smartphones.

However, we assume he is unaware of the deception mechanism, thus he can compromise and exploit any legitimate microservice and/or decoy in order to reach its targets. We model the attacker behavior within the microservice architecture as a sequence of lateral movements that are performed by means of two techniques:

- *Remote Code Execution (RCE)*: the attacker can violate a microservice by injecting malicious code throughout its interfaces [26]. The compromised microservice is used as foothold to violate other communicating microservices.
- *Container Escape*: the attacker exploits any non-root privilege capability configured in each container to run malicious code in other containers allocated on the same node [27]. In other words, the attacker can break the microservices logical isolation and violate non-communicating microservices within the same node.

Regardless of the employed technique, we assume that the attacker cannot accomplish a privilege escalation to gain access to the whole cluster (i.e. the full set of nodes running the containerized microservices). This attack scenario would bypass any deployed defense mechanism and it is mostly due to a poor configuration of the system privilege levels performed by the defender, which is beyond the deception scope.

B. Microservice architecture model

We assume that an organization provides some services leveraging a microservice architecture composed by $\{1, \dots, M\}$ microservices. Every microservice is virtualized and run on container-based technologies such as Docker. The organization has access to $\{1, \dots, N\}$ cloud computing nodes of fixed capacity in term of processing and memory capabilities denoted as C_n^{cpu} and C_n^{ram} , $n \in N$, respectively. Note that we do not differentiate between bare-metal nodes and virtual nodes. In other words, the hosted microservices can either run on virtual nodes in a public cloud, on bare-metal nodes in a private cloud model or on a mixture of the former two in a hybrid cloud model.

The microservices share the available resources and are allocated on the N computing nodes in order to fulfil the service level agreement requirements of the application. The microservice scheduling policy is handled by some orchestration tools and we assume that the resulting microservice configuration can be accessed by the proposed deception scheme. More precisely, the decoy allocation is computed according to the current microservice resource deployment that indicates the node and resource assignment for each microservice. Formally, we denote the amount of requested CPU and RAM resources by microservice m as $\bar{r}_m^{(cpu)}$ and $\bar{r}_m^{(ram)}$, respectively. We analytically represent the deployment of each microservice on a specific node by means of the indicator variables $r_{m,n}^{(cpu)}$ and $r_{m,n}^{(ram)}$, which express the number of CPU cycles and RAM volume consumed by microservice m running on node n . In particular, if microservice m is assigned to node n , then $r_{m,n}^{(cpu)} = \bar{r}_m^{(cpu)}$ and $r_{m,n}^{(ram)} = \bar{r}_m^{(ram)}$. Otherwise, if microservice m is not assigned to node n , then $r_{m,n}^{(cpu)} = 0$ and $r_{m,n}^{(ram)} = 0$. Based on this notation, we define the microservice resource deployment matrix as

$$\mathbf{R}^{(\cdot)} = [r_{m,n}^{(\cdot)} : \sum_{m \in M} r_{m,n}^{(\cdot)} \leq C_n^{(\cdot)} \quad \wedge \quad \sum_{n \in N} r_{m,n}^{(\cdot)} = \bar{r}_m^{(\cdot)}, \forall m \in M, \forall n \in N], \quad (1)$$

where the (\cdot) notation indicates that the above formulation holds both for the CPU resources and RAM resources. The first condition in (1) ensures the total number of resources requested by each microservice cannot exceed the node capacity. Similarly, the second condition guarantees that the microservice deployment is feasible by uniquely assigning a single node to each microservice.

C. Decoy configuration and placement model

The defender employs microservices as decoys in order to intercept ongoing cyber attacks targeting the deployed microservices. Each decoy can be considered as a clone of production microservices replicating the related functionalities as well as the interfaces towards adjacent communicating microservices. We assume that each decoy is deployed on the same node of the original cloned microservice. Although this assumption restrict the possible decoy placement strategy,

from a security perspective this requirements negates the addition of new container escape threats that are generated by allocating a decoy on a different node. In Fig. 2, we show an example of decoy placement that can be easily extended to any microservice deployment topology. Furthermore, every decoy is configured to provide the following features:

- *Attack detection reliability*: any data traffic intercepted by a decoy is considered as malicious since the legal data traffic is exclusively forwarded to production microservices. For example, this behavior could be implemented by defining specific network policies with Kubernetes [28]. This tool makes it possible to steer the application traffic flow among legitimate microservices while preserving their interfaces towards the decoys. Based on this configuration, the attacker can still interact with the decoys as the latter are only isolated from the application traffic and not from networking reachability.
- *High interactivity*: each decoy reacts to the attacker input like a production microservice. However, any data extracted by the attacker is fake and its content is configured to resemble the structure of production data (for example, a database filled with fake entries).
- *TTPs monitoring*: each decoy implementation is augmented by monitoring functionalities that allow the defender to gather cyber threat intelligence information on the attacker TTPs employed to violate the decoy.

The advantage of the proposed decoy configuration is twofold. It reduces the likelihood that the attacker can spot the deception mechanism as decoys are embedded within the microservice deployment topology like production microservices. It limits the decoy management complexity by duplicating microservices that are currently running.

Instead of cloning the up-to-date implementation of a production microservice, a deceptive microservice could employ a legacy implementation version characterized by known and non-patched vulnerabilities. On one hand, this design choice increases the decoy luring capability. On the other hand, beside possible compatibility issues with the current microservice deployment, it enlarges the attack surface of the infrastructure by intentionally introducing additional weaknesses that offer new TTPs to the attacker. For this reason, we do not consider such decoy configuration option and we plan to better investigate such tradeoff in a future work to improve the proposed decoy allocation strategy.

We indicate the set of allocated decoys as $D = \{d_m^{(j)}, m \in M, j \in \mathbb{N}_{>0}\}$, where $d_m^{(j)}$ refers to the j -th decoy replicating microservice m . Due to the resource limitation, a microservice can be replicated as decoy only if the resources reserved to the deception mechanism satisfies the same CPU and RAM resource request associated to the legitimate microservice. Therefore, assuming that microservice m is allocated on node n , we express a feasible decoy allocation as

$$\mathbf{x} = [x_m : x_m \bar{r}_m^{(\cdot)} \leq \delta \cdot \Delta C_n^{(\cdot)}, \forall n \in N], \quad (2)$$

where $x_m = |\{d_m^{(j)}\}|$ is the total number of decoys cloning microservice $m \in M$, $\delta \in [0, 1]$ is denoted as resource decoy ratio and indicates the share of available resources that

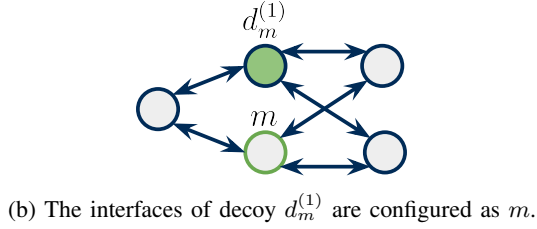
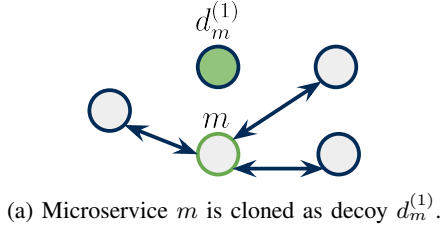


Fig. 2: Example of deceptive microservice allocation within a microservice architecture.

are reserved for the decoy allocation, $\Delta C_n^{(\cdot)}$ is the amount of unused resources in each node given the microservice resource deployment $\mathbf{R}^{(\cdot)}$. The condition in (2) ensures that the total resource request of decoys cloning a specif microservice can be accommodated by the available resources in the node. We assume that δ is fixed and its value is selected by the organization in order to balance the fulfilment of the service requirements and the effectiveness of deception mechanism. In general, the higher is the value of δ , the higher is the number of deployable decoys as more resources are dedicated for the deception mechanism. By computing the decoy allocation strategy according to a tunable share of the remaining resources, we ensure to not always require the full nodes capacity. This design choice mitigates the impact of the deception scheme on the production microservices whose service provisioning performance should always be prioritized.

D. Attacker model

We model the attacker capabilities using graph theory in order to better tailor the design of an effective decoy allocation strategy [29]. This mathematical tool can efficiently model the attacker lateral movements among microservices and decoys by abstracting the attacker techniques in the form of a sequence of graph vertices and edges. In detail, we introduce the *Attack Graph* (AG) associated to the current microservice deployment configuration as a directed acyclic graph $G = (V, E, f)$ where:

- $V = \{v_i \in M \cup D\}$ is the set of vertices in the graph corresponding to the active microservices and decoys.
- $E = \{(v_i, v_j) : v_i \prec v_j \wedge v_i, v_j \neq v_i \in V \times V\}$ is the set of directed edges identified by the ordered vertex pairs $e_{ij} = (v_i, v_j)$ that represent the microservices downstream call flow (in other words, we consider that microservice i requests a task to microservice j). More specifically, the vertex pair (v_i, v_j) is connected by the directional edge e if an attacker can move laterally from microservice i to microservice j by either leveraging RCE or container-escape techniques. Note that in the latter

scenario, microservices i and j must be allocated on the same computing node.

- $f : \{E \rightarrow w\}$ is a function assigning a weight $w \in \mathbb{R}^+$ to each directed edge e_{ij} expressing the microservice vulnerability level. Note that the same reasoning applies to decoys. In order to formalize this concept, we employ an approach similar to the one adopted by the authors of [30] that leverages the *Exploitability Metrics* (EM) and *Exploit Code Maturity* (ECM) indicators. The former measures the difficulty to exploit a vulnerability of a microservice, whereas the latter indicates the likelihood of that vulnerability being attacked [31]. Formally, we compute the weight for each edge e_{ij} as

$$f(e_{ij}) = \frac{\sum_{\nu \in \mathcal{V}_j} EM_\nu \cdot ECM_\nu}{\sum_{\nu \in \mathcal{V}_j} ECM_\nu} \quad \forall i \in V \quad (3)$$

where \mathcal{V}_j is the set of vulnerabilities associated to microservice $j \in M$. Practically, each vertex provides the weight w of its in-ward edges as a weighted average of the vulnerabilities associated to the microservice technical implementation. The lower is the value of w , the weaker is the security level of a microservice. The above formulation is extended to include virtualization vulnerabilities when both i and j are allocated on the same node.

Based on the definition of AG, we assume that the attacker reaches the target by violating the sequence of microservices providing the "least impedance" path in term of exploitability complexity given its entry-point. We quantify such attacker behavior as the shortest path in AG between a source vertex $v_s \in V$, that identifies the attacker entry-point, and a target vertex $v_t \in V$, that identifies the target microservice whose violation makes it possible for the attacker to access some organization assets. Formally, we define the *Attack Path* (AP) between v_s and v_t as the sequence of vertices

$$AP(v_s, v_t) = \{(v_s, \dots, v_i, \dots, v_t) : (v_i, v_{i+1}) \in E, \forall v_i \in V\} \quad (4)$$

that minimizes the total weight $\sum_{e \in AP(v_s, v_t)} f(e)$. Intuitively, each $AP(v_s, v_t)$ represents the worst-case attack scenario where a malicious user employ the most efficient sequence of techniques to penetrate the system. Moreover, if an AP contains at least one decoy between v_s and v_t , we denote this path as a *Deceptive Attack Path* (DAP), i.e.

$$DAP(v_s, v_t) = \{AP(v_s, v_t) : AP(v_s, v_t) \cap D \neq \{\phi\}\} \quad (5)$$

The amount of DAPs in AG measures the deception quality of the decoy placement configuration since the higher is this value, the higher is the number of attacks that are likely to be intercepted by the decoys.

IV. PROBLEM FORMULATION

Given the assumptions about the defender unawareness of the attacker entry point and targets, an effective decoy allocation maximizes the number of generated DAPs between any vertex pair in AG according to the resource availability.

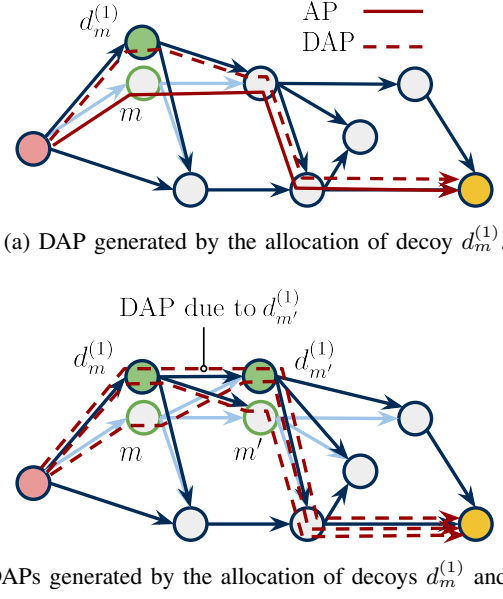


Fig. 3: Example of DAPs generated by a different number of decoys along a specific AP in AG. The red and yellow microservices represent the AP source and target, respectively.

We illustrate this intuition by analyzing the impact of the decoy allocation on the AG topology in Fig. 3. The latter depicts an example of DAP generation in a simple scenario but it can be easily extended for an arbitrary number of decoys and APs. Each decoy placement modifies the AG structure by adding a new “deceptive” vertex having the same features as the one corresponding to the replicated microservice. As a consequence, a decoy generates a number of DAPs equals to the number of APs traversing the cloned microservice (Fig. 3a). Note that those APs may also include DAPs that are introduced by other allocated decoys as highlighted in Fig. 3b, where decoy $d_m^{(1)}$ now generates an additional DAP due to the allocation of $d_{m'}^{(1)}$ on the same AP (the same argument applies for the additional DAP traversing $d_{m'}^{(1)}$).

Following these observations, the allocation of decoys on microservices sharing multiple APs can produce a high number of DAPs and thus increases the decoys likelihood to misdirect and intercept possible attacks. In other words, using an analogy, decoys can be considered as “magnets” attracting APs towards their location, hence an effective decoy placement is the one maximizing the total number of intercepted APs so to maximize the number of generated DAPs.

We formalize the decoy allocation dynamic by leveraging the concept of *Betweenness Centrality* (BC) of a vertex. This metric measures the number of shortest paths traversing each graph vertex and offers an analytic approach to tackle the decoy allocation problem. The higher this value is, the more “central” is the related vertex compared to the surrounding vertices since its location allows to reach multiple destinations with minimum cost. In our scenario, we define $\sigma_{AP(v_s, v_t)}^{(v_i)}$ as the binary indicator function expressing if the AP from v_s to v_t contains vertex v_i . More specifically, $\sigma_{AP(v_s, v_t)}^{(v_i)} = 1$ if $v_i \in AP(v_s, v_t)$ and $\sigma_{AP(v_s, v_t)}^{(v_i)} = 0$ otherwise. Similarly, we

define $\sigma_{AP(v_s, v_t)}$ as the variable indicating the number of APs from v_s to v_t . According to this notation, we compute the BC of a vertex $v_i \in V$ given the number of APs in AG as

$$b(v_i) = \sum_{v_s \neq v_i \in V} \sum_{v_t \neq v_s \in V} \frac{\sigma_{AP(v_s, v_t)}^{(v_i)}}{\sigma_{AP(v_s, v_t)}}. \quad (6)$$

The denominator of (6) can be considered as a normalization factor balancing the BC gain contribution of the multiple equal-cost DAPs generated by the decoys allocated on the same microservice.

A. Optimal decoy allocation

In practice, $b(v_i)$ allows to formally quantify the deception effectiveness of each decoy according to its placement. However, the formulation (6) cannot be employed as optimization objective since it is not expressed as a function of the number allocated decoys, which is indeed the optimization variable. In general, the computation of the BC of a vertex given an arbitrary number of graph modifications (e.g. vertex addition/removal and edge addition/removal) is a very challenging task since the corresponding shortest paths modifications can hardly be predicted [32].

However, by leveraging the AG topology structure as well as the decoy allocation dynamic, we can overcome this issue and propose an alternative formulation of $b(v_i)$ fulfilling the aforementioned requirement. In detail, we employ the initial number of APs going through each vertex in the original AG without decoys as a basis to update the BC value of any vertex when a decoy allocation is computed. As a matter of fact, the decoy allocation, which corresponds to a vertex addition operation, preserves the acyclic property of AG since the new introduced edges have the same direction of the in-ward and out-ward edges of the cloned microservice (see Fig. 3). This particular feature can be used to exactly predict the number of DAPs generated by any decoy placement configuration. Formally, we compute the BC of a vertex $v_i \in V$ given a feasible decoy allocation \mathbf{x} as

$$\bar{b}(v_i) = \sum_{v_s \neq v_i \in V} \sum_{v_t \neq v_s \in V} \bar{\sigma}_{AP(v_s, v_t)}^{(v_i)} \cdot \frac{(1 + x_s) \cdot (1 + x_t)}{(1 + x_i)}, \quad (7)$$

where $\bar{\sigma}_{AP(v_s, v_t)}^{(v_i)} = 1$ indicates the AP going through vertex $v_i \in M$ in the AG when no decoys are allocated, $\bar{\sigma}_{AP(v_s, v_t)}^{(v_i)} = 0$ otherwise. The numerator of (7) provides the total number of DAPs traversing vertex v_i that are generated from the vertex pairs (v_s, v_t) having x_s and x_t decoys allocated, respectively. Note that DAPs generated by vertex pairs not containing v_i are excluded from the computation since $\bar{\sigma}_{AP(v_s, v_t)}^{(v_i)} = 0$ by construction. In practice, the total number of DAPs is computed by summing up the individual contribution of the possible vertex pairs formed between the sets $\{v_s, v_{d_s^{(1)}}, \dots, v_{d_s^{(x_s)}}\}$ and $\{v_t, v_{d_t^{(1)}}, \dots, v_{d_t^{(x_t)}}\}$. Each vertex pair enumerates the DAP between the corresponding source and target vertices as $\sigma_{AP(v_j, v_k)}^{(v_i)} = 1$ where $j = \{s, d_s^{(1)}, \dots, d_s^{(x_s)}\}$ and $k = \{t, d_t^{(1)}, \dots, d_t^{(x_t)}\}$. The denominator of (7) accounts for the fact that any DAP generated by each vertex pair is also

replicated by decoys allocated on microservice i if $x_i > 0$, hence it acts as normalization factor like the denominator of (6).

Leveraging this alternative BC formulation, we compute the decoy allocation maximizing the the total number of DAPs given the microservice resource deployment configuration expressed by $\mathbf{R}^{(\cdot)}$ as

$$\max_{\mathbf{x}} \sum_{m \in M} x_m \cdot \bar{b}(v_m) \quad (8)$$

subject to

$$\sum_{m \in M} x_m r_{m,n}^{(cpu)} \leq \delta \cdot \Delta C_n^{(cpu)} \quad \forall n \in N \quad (9)$$

$$\sum_{m \in M} x_m r_{m,n}^{(ram)} \leq \delta \cdot \Delta C_n^{(ram)} \quad \forall n \in N \quad (10)$$

$$x_m \cdot (r_{m,n}^{(cpu)} - \bar{r}_m^{(cpu)}) \geq 0 \quad \forall m \in M, \forall n \in N \quad (11)$$

$$x_m \cdot (r_{m,n}^{(ram)} - \bar{r}_m^{(ram)}) \geq 0 \quad \forall m \in M, \forall n \in N \quad (12)$$

$$x_m \in \mathbb{N}, \quad \forall m \in M \quad (13)$$

Each addend in (8) provides the number of DAPs generated by the x_m decoys replicating microservice m . Constraints (9) and (10) ensure that the allocated decoys do not exceed the dedicated CPU and RAM resources defined by parameter δ , respectively. Constraints (11) and (12) enforce that each decoy must be allocated on the same node of the cloned microservice. Finally, constraint (13) expresses the integer nature of the considered allocation problem.

The proposed optimization problem is characterized by a non-linear objective function as it can be easily observed by the definition of $\bar{b}(v_i)$ in (7) which makes the solution computation very demanding. Moreover, the integrity constraint further exacerbates this issue as it renders the problem combinatorial. Intuitively, (8)-(13) can be seen as a Knapsack problem with variable coefficients $\bar{b}(v_i)$ and thus its complexity is NP-hard [33]. For this reason, this problem formulation is unpractical for real-world architectures having hundreds of active microservices whose resource occupation can be often reconfigured and thus leading to a decoy re-allocation. To overcome this limitation, we approximate the optimal allocation by designing an heuristic decoy allocation scheme to reduce the solution computational complexity.

B. Heuristic decoy allocation

The proposed heuristic consists of a greedy algorithm that prioritizes the allocation of decoys on microservices that require a low amount of resources and, at the same time, that are traversed by a high number of APs (in other words, it employs the BC values associated to each microservice in the original AG). In detail, the algorithm allocates one decoy at every iteration and update the DAPs generation according to the previously allocated decoys. This procedure makes it possible to progressively enumerate every DAP introduced by the decoys and thus can help approximating the optimal allocation. We present the pseudo-code for this scheme in Algorithm 1.

Algorithm 1 Heuristic algorithm

```

1: Input: AG,  $\bar{\sigma}$ ,  $b$ ,  $\mathbf{R}^{(\cdot)}$ ,  $\Delta C^{(\cdot)}$ 
2: Output:  $\mathbf{x}$ 
3: Initialize  $x_m = 0$  for each  $m \in M$ 
4: Initialize the priority queue  $Q$  to sort microservices
5: Compute  $\hat{d}_m = \lfloor \delta \Delta C_n^{(\cdot)} / \bar{r}_m^{(\cdot)} \rfloor$  for each  $m \in M$ 
6: for each  $m \in M$  do
7:   if  $\hat{d}_m \geq 1$  then
8:      $p \leftarrow -b(v_m) \cdot \hat{d}_m$ 
9:     Insert  $m$  into  $Q$  with priority  $p$ 
10:  end if
11: end for
12: while  $Q$  is not empty do
13:   Extract the first  $m$  from  $Q$ 
14:   Allocate a decoy on  $m$  and update the AG topology
15:   Update  $x_m \leftarrow x_m + 1$ 
16:   Update resource availability  $\delta \Delta C_n^{(\cdot)} \leftarrow \delta \Delta C_n^{(\cdot)} - \bar{r}_m^{(\cdot)}$ 
17:   for each  $i \in M$  do
18:     for each  $t \in M$  do
19:        $b(v_i) \leftarrow b(v_i) + (\bar{\sigma}_{AP(v_m, v_t)}^{(v_i)} + \bar{\sigma}_{AP(v_t, v_m)}^{(v_i)}) \cdot (x_t + 1)$ 
20:        $\theta_{v_t}^{(v_i)} \leftarrow \theta_{v_t}^{(v_i)} + (\bar{\sigma}_{AP(v_t, v_m)}^{(v_i)} + \bar{\sigma}_{AP(v_m, v_t)}^{(v_i)}) \cdot (x_m + 1)$ 
21:     end for
22:   end for
23:   Clear  $Q$ 
24:   Recompute  $\hat{d}_m$  for each microservice
25:   for each  $i \in M$  do
26:      $\Delta b(v_i) \leftarrow b(v_i) \cdot (x_i + 1) / (x_i + 2)$ 
27:     for each  $t \in M$  do
28:        $\Delta b(v_i) \leftarrow \Delta b(v_i) + [\sigma_{AP(v_m, v_t)}^{(v_i)} \cdot (x_t / x_t + 1)]$ 
29:     end for
30:      $p \leftarrow -\Delta b(v_i) \cdot \hat{d}_i$ 
31:     if  $\hat{d}_i \geq 1$  then
32:       Insert  $i$  into  $Q$  with priority  $p$ 
33:     end if
34:   end for
35: end while

```

Line 5 computes the number of deployable decoys \hat{d}_m on each microservice defined as the available resources on the assigned node divided by the microservice resource consumption. This value is used to weight each microservices in the priority queue Q according to the number of APs going through them in lines 8-9. We allocate a decoy on the microservice with the highest priority in Q (i.e. the one in the first position) by updating the AG topology in line 15 and we recompute the available resources for decoys in line 16. We iteratively calculate the number of generated DAPs in lines 17-22. In detail, line 19 updates the BC values for other microservices in AG given the new decoy allocation, while line 20 takes into consideration the total DAPs, denoted as $\theta_{v_t}^{(v_i)}$, generated by the interaction with the newly allocated decoy and the previously allocated ones. Lines 23-24 clean the queue and recompute the number of deployable decoys. These steps are needed to discard microservices that cannot be cloned

as decoys, thus they are no longer considered as possible candidates. Lines 26-28 weight the microservices priority according to the potential number of DAPs, expressed as the BC increment $\Delta b(i)$, that can be generated by allocating an additional decoy on the corresponding microservice. These steps guide the decoy allocation computation in the next iteration as the algorithm is incentivized to select microservices with the highest BC gain. Lines 31-33 insert the microservices into the queue with the updated priority only if the resources available are sufficient. The algorithm convergence is completed when Q is empty, which indicates that the resources reserved for the decoys are exhausted.

The overall algorithm time complexity can be computed as follows. We restrict the analysis for a single iteration, referred by lines 13-34, in order to provide the time complexity required to allocated a single decoy into the infrastructure. The most computationally expensive operation consists of the nested loops calculating the number of additionally DAPs generated by the interactions with previously allocated decoys. In this scenario, assuming that the priority queue Q is implemented using a binary heap, these steps have a complexity of $\mathcal{O}(M \cdot (M + M) + M \cdot (M + \log M))$, which can be further simplified to $\mathcal{O}(M^2)$. Since the total number of the algorithm iterations depends on the number of allocated decoys D , the overall algorithm complexity is $\mathcal{O}(D \cdot M^2)$.

V. PERFORMANCE EVALUATION

We evaluate the performance of the presented decoy allocation scheme by assessing the results with respect to a variable number of active microservices as well as resource decoy ratio. We compare the performance of the *Optimal* decoy allocation defined in (8)-(13) and the related *Heuristic* allocation presented in Algorithm 1 with the following three schemes.

- *Linear*: this scheme provides a decoy allocation that maximizes the number of DAPs without accounting for the additional paths introduced by the decoys. For example, with reference to Fig. 3, this scheme would ignore the DAP between $d_m^{(1)}$ and $d_{m'}^{(1)}$. In other words, this scheme neglects the AG topology modifications due to the decoy placement. The solution is computed by formulating a linear integer optimization problem having $\bar{\sigma}_{AP(v_s, v_t)}^{(m)}$ as objective function instead of (8). Formally, we have

$$\max_{\mathbf{x}} \sum_{m \in M} x_m \cdot \sum_{v_s \neq v_m \in V} \sum_{v_t \neq v_s \in V} \bar{\sigma}_{AP(v_s, v_t)}^{(v_m)} \quad (14)$$

subject to (9)-(13). Note that this formulation is still NP-hard as it can be reduced to the classical Knapsack problem. We assess the performance of this approach in order to analyze the performance gain provided by the optimal approach which models the decoy allocation dynamic over the AG.

- *Sidecar*: this scheme is a simple decoy placement strategy that allocates decoys on the most vulnerable assets. In our scenario, we identify the most vulnerable microservices as those ones having the lowest weight w associated to the in-ward edges in the AG. To provide a fair comparison

with other schemes which are resource-aware, we also account for the resource consumption of the decoys by selecting the microservices to be cloned according to the priority metric of the heuristic algorithm.

- *Random*: this scheme randomly allocate the decoys and it is used as performance lower bound.

A. Simulation setup

The simulation environment was developed in Python. In particular, the decoys deployment as well as the AG related computations were implemented using *NetworkX* library [34], which provides many functionalities to handle operations in graphs. The optimal and linear approaches were computed using the *SCIP* optimization suite [35].

Due to the impossibility to retrieve practical scenarios of large-scale microservice architectures, we synthetically generated the microservice diagram topology according to a *Barabasi-Albert* graph model [36]. The latter allows to generates graphs composed by few highly-connected vertices and many low-connected vertices. We employed this generation procedure as it resembles the structure of a limited number of microservices (for example, microservices working as front-ends) dispatching computational tasks toward multiple connected microservices that are rarely connected with one another.

To effectively model a realistic resource deployment of a microservice architecture, we employed a public dataset containing the traces of the normalized computing nodes capacities and the resource consumption of the allocated containers in the Alibaba cloud infrastructure [37]. We generated the resource deployment configuration matrices $\mathbf{R}^{(cpu)}$ and $\mathbf{R}^{(ram)}$ by randomly sampling a number of containers corresponding to the number of considered microservices. In other words, we mapped the resource consumption of Alibaba containers and the related allocation on the computing nodes as $r_{m,n}^{(ram)}$ and $r_{m,n}^{(cpu)}$. We chose the number of computational nodes N by progressively deploying each active microservice on the same node up to the 70% of its capacity before selecting a new one. For example, by following this procedure, the number of nodes required to accommodate 500 microservices, which is the highest microservice configuration considered in the simulation, is $N = 20$. Based on the resulting microservice deployment, we reserved a share of the remaining resources in each node for the decoys allocation according to δ .

Given the microservice diagram and the related microservice resource deployment, we extracted the AG as described in the previous section. We generated the edge weights of AG by first assigning a random number of vulnerabilities between 3 to 5 to each microservice, i.e. $|\mathcal{V}_m| \in \{3, 4, 5\}, \forall m \in M$. Then, for each vulnerability of every microservice, we extracted a random *EM* and *ECM* using the Common Vulnerability Scoring System Calculator tool [31] in order to compute the final value of w with (3).

Finally, we randomly sampled 100 configurations of microservice architectures and resource deployment and we plotted the average results within the 95% confidence intervals for each considered simulation instance.

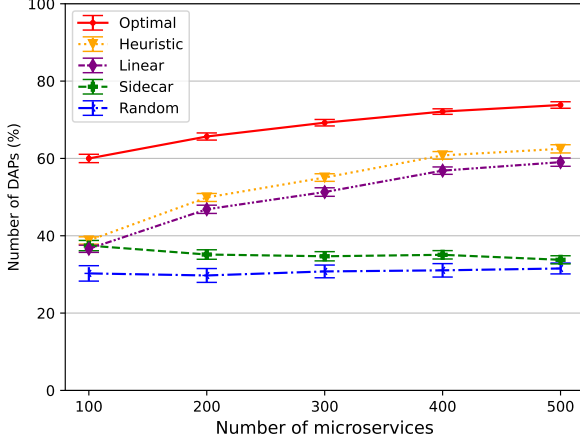


Fig. 4: Percentage of DAPs over the total number of APs when the number of microservices is increased from $M = 100$ to $M = 500$. The decoy resource ratio is $\delta = 0.3$.

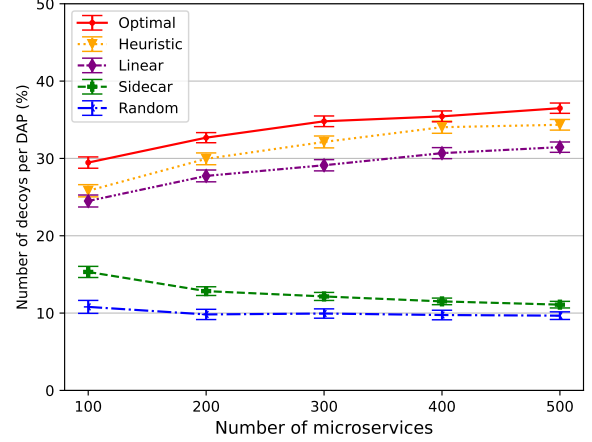


Fig. 5: Percentage of decoys per DAP when the number of microservices is increased from $M = 100$ to $M = 500$. The decoy resource ratio is $\delta = 0.3$.

B. Variable number of microservices

We assess the results by fixing the decoy resource ratio as $\delta = 0.3$ while increasing the number of microservice from $N = 100$ to $N = 500$.

In Fig. 4 we present the percentage of DAPs over the total number of AP in the AG. The optimal solution achieves the highest performance since it spreads more effectively the decoys within the microservice deployment topology. In detail, the optimal formulation selects microservices that are traversed by a high number of APs and, at the same time, that can be intercepted by a high number of DAPs. The heuristic scheme provides some notable optimality gap since its greedy approach prioritizes the allocation of decoys along the same AP instead of diversifying the allocation on other APs. However, it still generates a higher number of DAPs compared to the linear scheme. The latter underestimates the number of generated DAPs as it neglects the impact of the allocated decoy on the AG topology. This decoy placement configuration does not take advantage of DAPs generated between decoys to increase the possible number of intercepted attacks, hence it suffers from an evident performance loss compared to the optimal approach. The sidecar is outperformed by all schemes since it does not consider the sequence of microservices that must be violated by the attacker in order to reach the target. In detail, highly vulnerable microservices are reached by few APs when they are surrounded by more secure microservices. Consequently, the allocated decoys generates a lower number of DAPs. Moreover, by increasing the number of microservices, this effect is exacerbated as there are more microservices acting as “filters” that further reduce the number of APs traversing the selected microservices.

In Fig. 5 we show the average number of decoys per DAP in order to provide better insights on the deceptive capability of our scheme. Intuitively, the higher is the number of decoys per DAP, the higher is the probability that the attacker is going to interact with a decoy after each lateral movement.

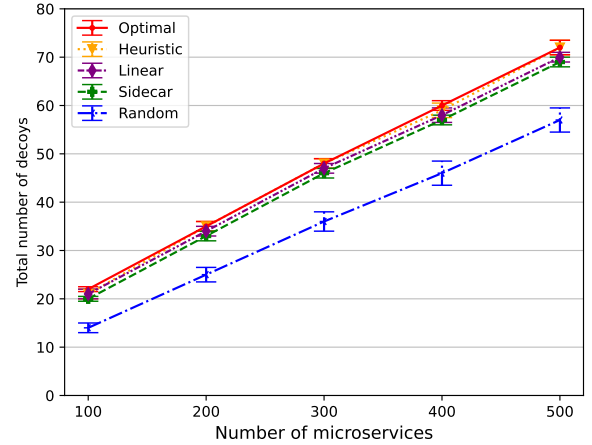


Fig. 6: Total number of allocated decoys when the number of microservices is increased from $M = 100$ to $M = 500$. The decoy resource ratio is $\delta = 0.3$.

In general, the discussion of the previous plot also applies for this scenario. The optimal scheme ensures the highest number of decoys per DAP as it places the decoys along microservices contained in multiple APs. The heuristic scheme achieves similar performance and provides better gain compared to the linear and sidecar schemes. In particular, the greedy nature of heuristic scheme incentives the allocation of decoys on microservices within the same AP. This strategy generates a high amount of DAPs at each new algorithm iteration thanks to the already allocated decoys along that AP in the previous steps.

In Fig. 6 we show the total number of allocated decoys. The number of decoys increases as more computing nodes are activated to accommodate the required microservices. Generally, excluding the random scheme which is resource-agnostic, the various schemes roughly deploy the same number

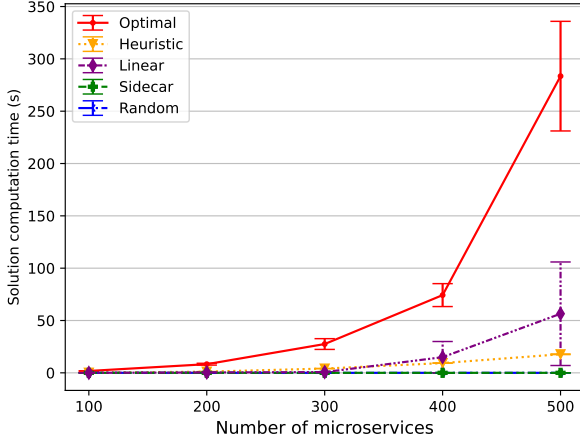


Fig. 7: Computational complexity of the considered schemes when the number of microservices is increased from $M = 100$ to $M = 500$. The decoy resource ratio is $\delta = 0.3$.

of decoys in a given microservice deployment configuration. This behavior suggests that the performance gain provided by the optimal and heuristic schemes derive from a decoy allocation that is tailored to the vulnerabilities of the current microservice deployment and it is not due to a higher number of allocated decoys that artificially inflates the number of generated DAPs. This statement is also supported by the following observation. Unlike the sidecar scheme, the modest decoy increment produced by the microservice scaling allows the optimal and heuristic scheme to further increase the number of DAPs. As a matter of fact, we highlight that when $N = 100$ that decoy-to-microservice ratio is roughly 20% instead when $N = 500$ is roughly 15%, meaning that the number of decoys increases way slower than the number of active microservices.

In Fig. 7 we show the convergence performance of the considered schemes. The non-linearity of (8) makes the optimal scheme unsuitable for large-scale microservice deployments that need to be frequently updated due to the high convergence time. Conversely, the heuristic decoy allocation requires a considerably lower computational complexity, thus it is preferable in the aforementioned scenarios. Moreover, the proposed heuristic also outperforms the linear approach, that suffers from a similar trend as the optimal scheme when the number of microservices is higher than 300. The sidecar scheme is characterized by a very low complexity due to the simplicity of its approach, which however achieve poor results in term of DAP generation as previously discussed. As a consequence, the optimal and heuristic schemes provides the most effective solutions to ensure an effective deception strategy and a scalable decoy placement configuration, respectively.

C. Variable ratio of dedicated deceptive resources

We also present the performance of the considered decoy allocation schemes for different configurations of available resource reserved for the deception mechanism. Specifically,

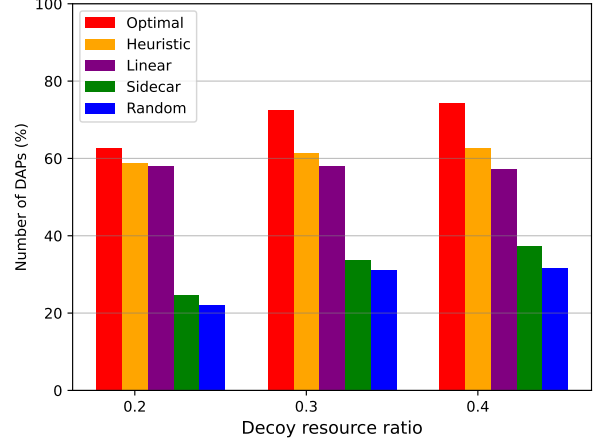


Fig. 8: Percentage of DAPs over the total number of APs when the decoy resource ratio is $\delta = \{0.2, 0.3, 0.4\}$. The number of active microservices is $N = 500$.

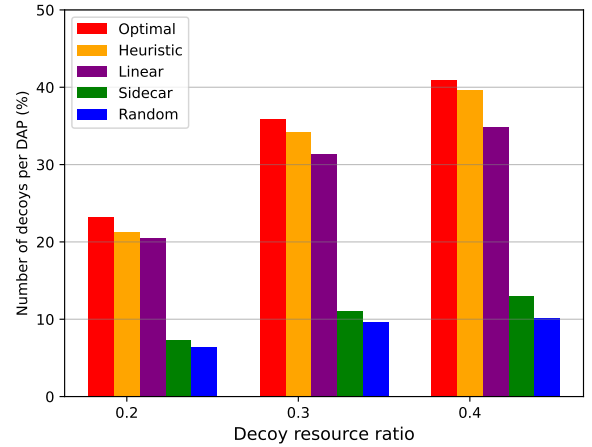


Fig. 9: Percentage of decoys per DAP when the decoy resource ratio is $\delta = \{0.2, 0.3, 0.4\}$. The number of active microservices is $N = 500$.

we restrict the analysis for $M = 500$ and we compute the decoy allocation when $\delta = \{0.2, 0.3, 0.4\}$.

In Fig. 8, we show the ratio of DAPs over the total number of APs for different resource configurations. By increasing the number of resources assigned for the decoy deployment, the optimal and heuristic schemes increase the APs coverage by placing a higher number of decoys. Conversely, we remark the performance degradation of the linear scheme whose underestimation error of the DAPs number gets more severe as the decoy deployment availability increases.

In Fig. 9, we show the average number of decoys per DAP. The heuristic scheme performance is comparable to the optimal scheme in every configuration. This result is achieved by distributing the decoy allocation on the available nodes in order to balance the resource consumption. In detail, by ranking microservices according to the number of deployable

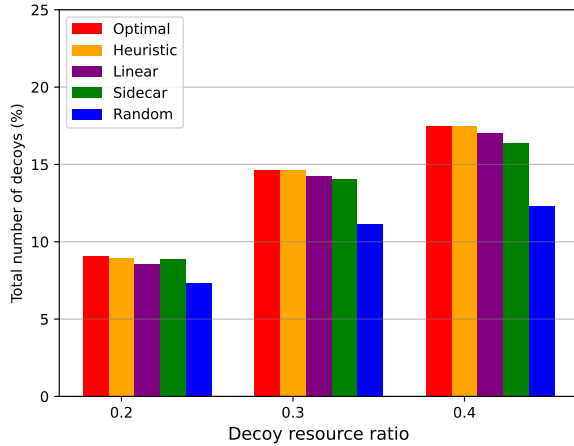


Fig. 10: Percentage of decoys over the total number of microservices when the decoy resource ratio is $\delta = \{0.2, 0.3, 0.4\}$. The number of active microservices is $N = 500$.

decoys, the heuristic scheme prioritizes the placement of decoys that can produce a high number of DAPs and at the same time, have a low resource consumption.

Finally, in Fig. 10 we shows the percentage of allocated decoys. As previously observed, all schemes deploy the same number of decoys in every resource configuration scenario. This fact further highlights the resource-efficiency of the proposed approach that leverages the topological structure of the APs to tradeoff the resource consumption for an effective decoy allocation.

VI. CONCLUSION

Cyber deception is a promising defense strategy to enhance the security of microservice architectures. By deploying deceptive microservices within the infrastructure, it is possible to intercept attacks and learn about attackers' TTPs with a limited complexity overhead. In light of this, we considered the problem of a resource-aware decoy allocation strategy under a limited budget of available computational resources. We modeled the attacker lateral movements between microservices using graph theory, where we considered an attack path as the sequence of microservices violated by the attacker in order to reach its target. We proposed an analytical formulation to quantify the number of deceptive attack paths generated by an arbitrary decoy placement, and used this expression to design an integer non-linear optimization problem that maximizes the number of deceptive attack paths given the available resources. We also designed a low-complexity heuristic decoy allocation scheme to approximate the optimal solution. In the performance evaluation, our approach outperformed benchmark schemes on several metrics while using the same number of decoys.

REFERENCES

- [1] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice architecture: aligning principles, practices, and culture*. "O'Reilly Media, Inc.", 2016.
- [2] A. Hannousse and S. Yahiouche, "Securing microservices and microservice architectures: A systematic mapping study," *Computer Science Review*, vol. 41, p. 100415, 2021.
- [3] S. Jajodia, V. Subrahmanian, V. Swarup, and C. Wang, *Cyber deception*. Springer, 2016, vol. 6.
- [4] L. Zhang and V. L. Thing, "Three decades of deception techniques in active cyber defense-retrospect and outlook," *Computers & Security*, vol. 106, p. 102288, 2021.
- [5] Q. Duan, E. Al-Shaer, M. Islam, and H. Jafarian, "Conceal: A strategy composition for resilient cyber deception-framework, metrics and deployment," in *2018 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2018, pp. 1–9.
- [6] K. Ferguson-Walter, M. Major, C. K. Johnson, and D. H. Muhleman, "Examining the efficacy of decoy-based and psychological cyber deception," in *USENIX Security Symposium*, 2021, pp. 1127–1144.
- [7] M. M. Islam, A. Dutta, M. S. I. Sajid, E. Al-Shaer, J. Wei, and S. Farhang, "Chimera: Autonomous planning and orchestration for malware deception," in *2021 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2021, pp. 173–181.
- [8] A. Nehme, V. Jesus, K. Mahbub, and A. Abdallah, "Securing microservices," *IT Professional*, vol. 21, no. 1, pp. 42–49, 2019.
- [9] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security: A research perspective," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.
- [10] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 243–246.
- [11] Q. Zhu, "Game theory for cyber deception: a tutorial," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, 2019, pp. 1–3.
- [12] A. H. Anwar and C. Kamhoua, "Game theory on attack graph for cyber deception," in *International Conference on Decision and Game Theory for Security*. Springer, 2020, pp. 445–456.
- [13] C. Wang and Z. Lu, "Cyber deception: Overview and the road ahead," *IEEE Security & Privacy*, vol. 16, no. 2, pp. 80–85, 2018.
- [14] M. Zhu, A. H. Anwar, Z. Wan, J.-H. Cho, C. A. Kamhoua, and M. P. Singh, "A survey of defensive deception: Approaches using game theory and machine learning," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2460–2493, 2021.
- [15] B. Götz, D. Schel, D. Bauer, C. Henkel, P. Einberger, and T. Bauernhansl, "Challenges of production microservices," *Procedia CIRP*, vol. 67, pp. 167–172, 2018.
- [16] T. Yarygina and A. H. Bagge, "Overcoming security challenges in microservice architectures," in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. IEEE, 2018, pp. 11–20.
- [17] A. Osman, P. Bruckner, H. Salah, F. H. Fitzek, T. Strufe, and M. Fischer, "Sandnet: towards high quality of deception in container-based microservice architectures," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.
- [18] L. Li, J. Wu, W. Zeng, and X. Cheng, "A cyber deception method based on container identity information anonymity," *IEICE Transactions on Information and Systems*, vol. 104, no. 6, pp. 893–896, 2021.
- [19] H. Li, Y. Guo, P. Sun, Y. Wang, and S. Huo, "An optimal defensive deception framework for the container-based cloud with deep reinforcement learning," *IET Information Security*, vol. 16, no. 3, pp. 178–192, 2022.
- [20] H. Li, Y. Guo, S. Huo, H. Hu, and P. Sun, "Defensive deception framework against reconnaissance attacks in the cloud with deep reinforcement learning," *Science China Information Sciences*, vol. 65, no. 7, pp. 1–19, 2022.
- [21] A. H. Anwar, C. A. Kamhoua, N. O. Leslie, and C. Kiekintveld, "Honeypot allocation for cyber deception under uncertainty," *IEEE Transactions on Network and Service Management*, 2022.
- [22] J. Liu, Z. Wang, J. Yang, B. Wang, L. He, G. Song, and X. Liu, "Deception maze: A stackelberg game-theoretic defense mechanism for intranet threats," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.
- [23] A. H. Anwar and C. A. Kamhoua, "Cyber deception using honeypot allocation and diversity: A game theoretic approach," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2022, pp. 543–549.

- [24] H. Ma, S. Han, N. Leslie, C. Kamhoua, and J. Fu, "Optimal decoy resource allocation for proactive defense in probabilistic attack graphs," *arXiv preprint arXiv:2301.01336*, 2023.
- [25] M. A. R. A. Amin, S. Shetty, L. Njilla, D. K. Tosh, and C. Kamhoua, "Online cyber deception system using partially observable monte-carlo planning framework," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2019, pp. 205–223.
- [26] N. Mateus-Coelho, M. Cruz-Cunha, and L. G. Ferreira, "Security in microservices architectures," *Procedia Computer Science*, vol. 181, pp. 1225–1236, 2021.
- [27] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, vol. 7, pp. 52 976–52 996, 2019.
- [28] G. Budigiri, C. Baumann, J. T. Mühlberg, E. Truyen, and W. Joosen, "Network policies in kubernetes: Performance evaluation and security analysis," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2021, pp. 407–412.
- [29] S. Milani, W. Shen, K. S. Chan, S. Venkatesan, N. O. Leslie, C. Kamhoua, and F. Fang, "Harnessing the power of deception in attack graph-based security games," in *International Conference on Decision and Game Theory for Security*. Springer, 2020, pp. 147–167.
- [30] H. Jin, Z. Li, D. Zou, and B. Yuan, "Dseom: A framework for dynamic security evaluation and optimization of mtd in container-based cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 3, pp. 1125–1136, 2019.
- [31] C. S. I. Group, "Common vulnerability scoring system v3. 0: Specification document," 2019.
- [32] D. A. Bader, S. Kintali, K. Madduri, M. Mihail *et al.*, "Approximating betweenness centrality," in *WAW*, vol. 4863. Springer, 2007, pp. 124–137.
- [33] H. Suzuki, "A generalized knapsack problem with variable coefficients," *Mathematical Programming*, vol. 15, no. 1, pp. 162–176, 1978.
- [34] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.
- [35] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig, "The scip optimization suite 8.0," 2021. [Online]. Available: <https://arxiv.org/abs/2112.08872>
- [36] S.-H. Yook, H. Jeong, and A.-L. Barabási, "Modeling the internet's large-scale topology," *Proceedings of the National Academy of Sciences*, vol. 99, no. 21, pp. 13 382–13 386, 2002.
- [37] C. Lu, K. Ye, G. Xu, C.-Z. Xu, and T. Bai, "Imbalance in the cloud: An analysis on alibaba cluster trace," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 2884–2892.