MisMesh: Security Issues and Challenges in Service Meshes

Dalton A. Hahn(✉), Drew Davidson, and Alexandru G. Bardas

EECS Department, ITTC University of Kansas, Lawrence, KS, USA
{daltonhahn,drewdavidson,alexbardas}@ku.edu

Abstract. Service meshes have emerged as an attractive DevOps solution for collecting, managing, and coordinating microservice deployments. However, current service meshes leave fundamental security mechanisms missing or incomplete. The security burden means service meshes may actually cause additional workload and overhead for administrators over traditional monolithic systems. By assessing the effectiveness and practicality of service mesh tools, this work provides necessary insights into the available security of service meshes. We evaluate service meshes under skilled administrators (who deploy optimal configurations of available security mechanisms) and default configurations. We consider a comprehensive set of adversarial scenarios, uncover design flaws contradicting system goals, and present limitations and challenges encountered in employing service mesh tools for operational environments.

Keywords: Service mesh · DevOps · Containers · Consul · Istio · Linkerdv2

1 Introduction

The widespread enthusiasm of large enterprises for *microservice* system architectures [2], where many lightweight containers are managed and deployed via automation tools [20], lacks a matching evaluation of their security. A number of academic works have examined the security of individual containers [6,30,34]. However, the *service meshes* of interdependent microservices, are largely unstudied. Service meshes aid the microservice design philosophy of refactoring monolithic applications into distinct components that collaborate at scale [24].

Service meshes ease the complexity of managing microservice architectures by allowing the administrator to express the structure and relationships between services using configuration files [16,23]. State-of-art service mesh tools such as Consul [13], Istio [22], and Linkerdv2 [19] launch collections of microservices automatically. Furthermore, these tools automate *service discovery*, the process of locating and binding services together. Service discovery is a non-trivial process under the DevOps [2,3] ideology to support a range of flexible deployments. As such, service discovery is decentralized with dependencies satisfied dynamically.

N. Park et al. (Eds.): SecureComm 2020, LNICST 335, pp. 140–151, 2020.
https://doi.org/10.1007/978-3-030-63086-7_9

In studying service mesh security, we discover that misconfiguration issues and lack of security mechanisms enable numerous attacks. We view these as consequences of design flaws in service mesh security. When facing these attacks, service meshes either offer no defense or require significant manual intervention on the part of the system administrator. The latter effectively undermines a core goal of service meshes: ease of automation.

Current practices such as infinite-lifetimes and shared encryption keys [14] indicate that the design of service meshes has overlooked important security concerns. Nonetheless, deployment of immature service meshes is growing in production environments [5,9,10,25,39]. Moreover, the context-dependent scope and implementations of service meshes are so diverse that establishing a meaningful comparison between different tools is difficult.

Despite the building importance of defending service meshes, we are unaware of any systematic assessment of their security. To the best of our knowledge, this paper presents the first study to specifically focus on existing security mechanisms in service meshes. Our assessment indicates service mesh security implementations and maintenance mechanisms are incomplete, or even non-existent. Additionally, we discovered that even though service mesh tools advertise their security contributions, they are either not enabled by default, or are left to third-party tools to implement.
Our contributions can be summarized as follows:

- We present the first study to examine the security design and analyze the available security mechanisms within current service meshes
- We propose a relevant threat model to the service mesh domain and assess the effectiveness of existing tools to mitigate these threats
- We assess the impact and the effort of utilizing available security features in current service mesh tools

2 Background

Microservice architectures consist of a complex web of narrowly-scoped, interacting services in place of a monolithic architecture. This structure better enables incremental changes, resilience to cascading failures, and quicker update/release cycles [3,11,35] at the cost of complexity; it is a significant challenge to maintain synergy between services. Systems such as Kubernetes [26] provide a framework to deploy, scale, and manage microservices quickly, magnifying the need to coordinate services. Service meshes seek to address this gap between fast deployments and collaborating webs of microservices. In this section, we describe some of the enabling tools and design concepts that underlie service meshes.

Service Mesh Tools: Service meshes enable a service to be registered to a cluster, discovered dynamically by other dependent services, and to have configuration state maintained. Consul, Istio, and Linkerdv2 are the current state-of-art service mesh tools with full, production-ready releases. A major cause of complexity in coordinating services is to determine cluster membership and

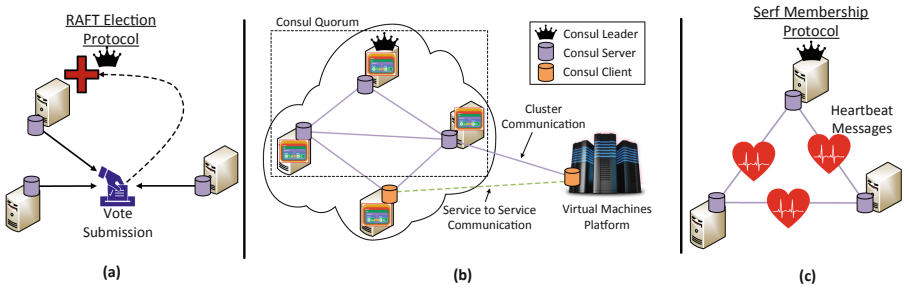

Fig. 1. Model Consul Service Mesh – Using Consul, the creation and operation of a model service mesh are shown. **(a).** RAFT elections occur periodically among Consul servers to determine cluster leadership. **(b).** Proxies present on each node route cluster- and service-level communications to nodes. Proxies may be installed on a variety of platforms including virtualized, containerized, and physical machines with little restriction on operating system [1]. **(c).** The Serf membership protocol occurs with high frequency to send heartbeat messages among nodes to track health and membership.

node operation status. Consul implements the Serf [17] membership and node health protocol (an extension of SWIM [8]) and the RAFT consensus protocol. The basic process is illustrated in Fig. 1. Leveraging the cluster membership logic, Consul creates a membership hierarchy to organize the permissions that members of the cluster possess to take action within the cluster. The Consul quorum is responsible for maintaining a consistent membership registry and holding cluster elections for the cluster permission hierarchy.

Istio and Linkerdv2 both require an underlying Kubernetes platform to provide cluster membership logic. In contrast, installation of Consul is supported on a range of operating systems and architectures as well as virtualized and physical instances [1]. Without a previously created and configured Kubernetes cluster of *pods*; collections of containers with shared resources [27,28], Istio and Linkerdv2 are unable to provide any of their promised features or security benefits. By imposing the initial requirement of a properly installed, configured, and secured Kubernetes infrastructure, in addition to the overhead of configuring and maintaining the service mesh, Istio and Linkerdv2 demonstrate a higher burden on system administrators than that of Consul. In contrast to Consul, Istio and Linkerdv2 do not maintain a hierarchical structure for permissions and state management, instead, they rely upon a star topology-like system where the Kubernetes master controls the cluster's pods either remotely, or locally, and sets the configuration and permissions of specific members within the cluster.

Service Mesh Security: The paradigm shift from monolithic systems to microservice systems has caused a change from *intra*-service issues to *inter*-service issues. This transitions the burden of security from within the operating system of a machine to across network connections. Issues previously addressable by trusted security measures within the operating system must now be addressed with network-level security measures. These issues include the need to protect

Table 1. Adversarial Goals on a Consul Deployment – Presents experimental results of achieved adversarial goals on a properly configured Consul service mesh deployment. **D**isruption: Interruption to service availability. **M**anipulation: Infiltration or exfiltration of data to cluster. **T**akeover: Adversary assumes the leadership position in cluster.

	Datacenter Label as a Secret	UDP Encryption	ACLs (Access Control Lists)	TLS Encryption	All Mechanisms Combined (Datacenter Label, UDP, ACLs, TLS)
Unprivileged Threat	D M T	—	D	—	—
Client Compromise	D M T	D M T	D	M	—
Server Compromise	D M T	D M T	D	M	—
Leader Compromise	D M T	D M T	D M T	D M T	D M T

cluster-level communications, service-level communications, and access permissions, both at the cluster-level as well as the service-level.

3 Threat Model and Experimental Design

To evaluate the security of modern service mesh tools, we used Consul as a model for service mesh design and implementation. We constructed a proof-of-concept environment using Consul to conduct our experiments. We consider the available security mechanisms for administrators and examine a deployment utilizing all available mechanisms as well as one using default configurations. Under these setups, we conduct a series of active attacks and report our results. We also present a comparison of available and default security mechanisms within Istio and Linkerdv2 and provide our findings. We utilize these findings to frame a discussion of the shortcomings and overhead system administrators should expect when attempting to secure service mesh clusters within their infrastructure.

Consul provides a meaningful representation of service meshes and the maturity of these tools. Of the current state-of-art service meshes, Consul is the most feature-rich and flexible tool available in this domain. As mentioned previously, Consul can be used with any other tools or forms of virtualization such as containers or virtual machines whereas Istio and Linkerdv2 are dependent upon an underlying Kubernetes implementation to provide necessary features for the mesh. Additionally, as of the writing of this work, Consul appears to be the most actively developed tool, enjoying the largest number of GitHub contributors (594) of the tools we encountered, and a comparable number of GitHub repository stars to the runner-up tool, Istio [15,21]).

Threat Model: The threat model we employ in this work considers common attacker goals of disruption of services and exfiltration of sensitive data. However, we also consider adversarial targets that are unique to the service mesh domain. For example, an attacker may often desire to infiltrate the cluster and gain privilege rather than destroying the functionality of a system. By infiltrating the cluster, the attacker may inject malicious service configurations to possibly redirect benign service requests to externally controlled endpoints. In Table 1, we denote these high-level goals as **D**isruption, **M**anipulation, and **T**akeover

for disruption to services and cluster activities, tampering of sensitive data via manipulation, and gaining privilege through service mesh takeover, respectively.

Experimental Setup: We deployed our model cluster upon a Dell R540 server configured with 128 GB of RAM, Xeon Gold 5117 processor, and 10 TB of SSD storage. We believe this hardware to be comparable to what would be utilized in production environments, both in on-site and remote, cloud datacenters.

The proof-of-concept Consul service mesh consists of an initial leader node or "bootstrapper" responsible for initializing the cluster and connecting the initial nodes. Alongside the leader node are two server nodes, forming the quorum, and a singular client node. Using Fig. 1 as our model, we manually deployed and configured these four Consul nodes (one leader, two servers, and one client node). We utilize only one client node due to the equivalent functionality of subsequent clients. Due to the architecture of Consul service mesh clusters, it is recommended to have 3 nodes acting as servers (one leader node and 2 server nodes) to manage the quorum and maintain the cluster state and log files [14]. Nodes are the main structural components of service mesh clusters, hosting ephemeral or long-lived microservices on permanent, virtual, or physical instances.

4 Evaluation of Modern Service Meshes

We consider an administrator with deep knowledge of the employed tool and its available security mechanisms. We present an experimental assessment of this "idealized" scenario and compare the results against default offerings of the tool.

With deep knowledge of available security mechanisms and their correct configuration, an administrator can leverage these protections to their greatest potential. To study how varying degrees of attacker strength can affect the level of compromise under these security mechanisms, we position the adversary at different levels of initial compromise. The lowest initial power we consider an attacker to have is that of an "Unprivileged Adversary" who has not yet compromised any node within the cluster. The highest initial level of power we consider is that of "Leader Compromise" where an adversary has the preliminary position of a node considered to be the leader of the Consul quorum. Under the assumption of a knowledgeable administrator and the preconditions placed upon the adversary, we evaluate the experimental results and provide our assessment.

Consul – Datacenter Label as a Secret: The first means of potential defense we consider within our proof-of-concept Consul service mesh is the datacenter label. We consider this a potential security mechanism due to the fact that if a prospective cluster node is configured with a datacenter label that differs from the target cluster, the prospective node will be denied membership to the cluster.

As shown in Table 1, under all adversarial scenarios, using strictly "Datacenter Label as a Secret" is insufficient in thwarting attacks against the cluster. Specifically, when using datacenter label alone, communication messages are exchanged in plaintext between nodes of the cluster. Due to the realistic possibility of an adversary to capture a *single* packet exchanged between the nodes

of the cluster, they may extract the datacenter label from the packet. The malicious join operation is, subsequently, made possible and once a member of the cluster, all high-level attacker goals can be achieved.

Consul – UDP Message Encryption: Next, we consider the Consul service mesh deployed using UDP message encryption as the sole mechanism of defense. As shown in Table 1, enabling UDP message encryption thwarts an unprivileged adversary from achieving any of their goals, but fails to provide protection under compromise of any cluster members. By enabling UDP encryption, the adversarial joins previously possible are prevented because an attacker is unable to decrypt packets from the legitimate nodes.

All nodes within a Consul service mesh share the same encryption key. To exacerbate this concern, Consul, as of the writing of this work, fails to provide any means of key revocation or rotation. In order to provide key rotation within the cluster, even through a separate "recovery" mechanism such as an SSH [36] session, all nodes must be stopped, configurations adjusted, and the cluster recreated. While the managed services of the cluster may be transient and possibly short-lived, the underlying service mesh infrastructure is intended to be long-living. Therefore, support for key rotation capabilities is vital for managing and maintaining a secure service mesh architecture.

Consul – ACLs: As shown in Table 1, Access Control Lists (ACLs) are highly effective at thwarting the adversarial goals of manipulation and takeover within the cluster. However, ACLs prove futile against disruption of cluster activities and service availability. In order for a system administrator to enable ACLs as a defense mechanism, extensive permission policies must be created and access tokens exchanged using a third-party, secure channel such as SSH. With a lack of support for distributing security objects safely within Consul itself, the implementation of ACLs, and subsequently the policies and tokens generated, demonstrates that security mechanisms within service meshes have been "bolted-on" to existing software, rather than incorporated into system design.

In order to secure the simple, four node service mesh used for our evaluation, as advised by the Consul tutorials [14], an administrator would need to generate unique access policies, generate tokens, and distribute and assign the generated tokens to proper recipients. All of these actions must be conducted from the single leader node due to the advised "operator-only" policy. Under the "operator-only" policy, permissions to edit the ACLs are restricted to the leader, meaning a singular node is responsible for all creation and distribution of policy materials. In direct contrast to the decentralized, distributed nature of the service mesh, the security structure implemented has been consolidated to a single point of control, the Consul leader. Augmenting the burden placed upon system administrators, the current implementation of Consul ACLs have no token rotation policy in place. Therefore, either the created access tokens within the cluster will exist for the lifetime of the cluster, or are revoked after a period of time, but with no means of redistributing fresh tokens to nodes.

When ACLs are the sole mechanism of defense for a Consul service mesh cluster, they prove ineffective at mitigating adversarial disruption efforts. Due

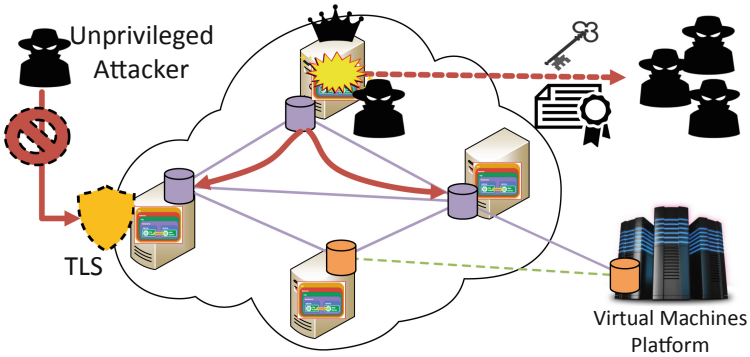

Fig. 2. TLS Message Encryption – Encrypting service traffic with TLS prevents an unprivileged attacker influencing the cluster. However, a leader node compromise allows an adversary to generate malicious TLS key pairs and exfiltrate them to other adversary nodes. Once additional adversaries join they may join the quorum and cast votes due to their server-permissioned certificates.

to implementation of processing access control policies within the service mesh, unauthorized messages must be confirmed as illegitimate by the cluster. Using around 25 adversarial nodes, we were able to disrupt operations within the service mesh by overwhelming the consensus protocol.

Consul – TLS Message Encryption: In order to protect service-level communication within the Consul cluster, a system administrator may enable TLS message encryption. To provide nodes the capability to sign messages, they must first have signed certificates from the certificate authority. When constructing the service mesh, the administrator would create a certificate authority from one of the server nodes of the cluster. Afterwards, the certificate authority is responsible for generating all server and client certificates. Distribution of certificates must be completed through a separate channel before cluster creation.

By enabling TLS encryption, the unprivileged adversary is unable to maliciously join the cluster, preventing any goals from being achieved in this case. Despite this, there are no protections for the key/value storage system. Accessing the key/value storage allows an adversary to manipulate configurations or secrets stored within the cluster. Figure 2 shows how, should the leader node ever be compromised in the lifetime of the cluster, an adversary may leverage the signing privileges of the certificate authority to generate illegitimate certificates and keys for malicious nodes.

The implementation of the certificate hierarchy within Consul once again shows a disconnect between the desired decentralized and distributed nature of service meshes with a centralized, consolidated security structure. Within Consul, the only node able to sign certificates of any privilege is the certificate authority (commonly created on the quorum leader node). Additionally, should the leader node fail, barring replication of the certificate authority key, the cluster

loses the ability to sign new certificates, once again conflicting with the flexibility goal of the DevOps ideology. Lastly, a lack of revocation and rotation mechanisms within Consul itself necessitates a third-party tool such as HashiCorp's Vault [18] or SSH be used to distribute fresh certificates to nodes, which triggers the need for widespread edits to configurations in order to return to a secure state.

Consul – All Mechanisms Combined: By enabling and combining all available security mechanisms, Table 1 shows a clear improvement in mitigating adversarial goals. However, employing all mechanisms presents administrators with a daunting amount of manual configuration. Considering the cost required to establish a secure model example with trivial functionality, the requirements to successfully deploy and secure enterprise-level systems is unreasonable. Also, due to the implementation of the available security mechanisms, should the leader of the cluster ever be compromised across the lifetime of a cluster, all configuration effort must be repeated to redeploy securely. By lacking necessary revocation and rotation mechanisms, Consul has limited the ability to construct dynamic service mesh clusters that are resilient to compromise events. Service mesh tools, while aiming to fill the niche of microservice architecture discovery, connection, and management, may, in fact, lead to substantial overhead for administrators who wish to deploy these tools in a secure fashion.

Nacos: While not directly considered a service mesh, Nacos provides many of the same features as the service meshes considered and has the ability to be configured in a way to accomplish many of the same high-level goals as service mesh tools. However, it is important to note that Nacos is technically a service discovery and management tool. As of the writing of this work, Nacos is in version 1.1.4, and is available for public use. However, Nacos has very little, if any security mechanisms available to its users. In its current state, Nacos depends primarily upon external security mechanisms such as firewalls, subnetting, and other perimeter defenses for protection.

Default Security Mechanisms: Table 2 outlines the available security mechanisms of service meshes and Kubernetes and the default state of these mechanisms. While Consul offers all of the necessary security capabilities to administrators, it *fails* to enable any of them by default and lacks rotation support for all mechanisms. With an extensive list of configurations to create and assign, such as node permissions, key creation and distribution, and certificate hierarchy, the overhead for system administrators is significant.

Additionally, Istio and Linkerdv2 fail to provide means of securing the cluster-level functionality and service-level access control by default. However, Linkerdv2 does enable service-to-service message encryption via mutual TLS by default, representing a valuable design decision that benefits security. In Istio, to provide the same service-level security, an administrator would be required to modify configurations of the cluster and provide additional authentication rules for individual pods and services in order to provide proper, secure functionality.

Table 2. Security Mechanisms in Service Mesh Tools – A summarized view of the security mechanisms available in each service mesh tool analyzed, which mechanisms are enabled by default, and additional details about the actual implementations. *Pod-to-pod encryption left to third-party implementation [27]. **Inherited from Kubernetes' Role-Based Access Control system [19,23].

Tool	Security Mechanism	Available in Tool?	Enabled by Default?	Default Lifetime	Revocation	Redistribution
Consul	Cluster Message Encryption	Yes	No	∞	No	No
	Service Message Encryption	Yes	No	1 year	Yes	No
	Cluster Access Control	Yes	No	∞	Yes	No
	Service Access Control	Yes	No	∞	Yes	No
Linkerdv2	Cluster Message Encryption	No	No	N/A	N/A	N/A
	Service Message Encryption	Yes	Yes	24 hours	Yes	Yes
	Cluster Access Control	No	No	N/A	N/A	N/A
	Service Access Control	Yes	No	∞**	No**	No**
Istio	Cluster Message Encryption	No	No	N/A	N/A	N/A
	Service Message Encryption	Yes	No	Ext Tool [23]	Ext Tool [23]	Ext Tool [23]
	Cluster Access Control	No	No	N/A	N/A	N/A
	Service Access Control	Yes	No	∞**	No**	No**
Kubernetes	Cluster Message Encryption	No*	No	N/A	N/A	N/A
	Service Message Encryption	Yes	No	1 year	Beta	Beta
	Cluster Access Control	Yes	No	∞	No	No
	Service Access Control	Yes	No	∞	No	No

5 Related Work

To our knowledge, this work is the first systematic study of service mesh security. Many of our attacks are inspired by existing work, and many of the implications of our work build upon previous studies of microservice security and networked systems. We discuss some works most closely related to our own below.

Microservice Security: Automation and the decentralized nature of microservice security has been observed or utilized by a number of previous works. Rastogi, *et al.* [31] evaluate an automation system for dismantling a monolithic software deployment into a collection of collaborating microservices in order to better adhere to the principle of least privilege [33]. Yarygina, *et al.* [40] note the comparative lack of security protections for Docker containers, and propose a container security monitor. In Sun, *et al.* [38], the authors study how the trust relationship between deployed microservices may result in the compromise of an entire system and they propose a system for deploying network security monitors in microservice environments to detect and block threats to clusters.

A number of previously published works focus on the security of individual Docker containers, which are frequently used for microservices. A representative example is Enck, *et al.* [34], which studies the risk of deploying containers automatically from 3rd-party container repositories. In Lin, *et al.* [29] and Martin, *et al.* [30], the authors examine attacks and countermeasures to the security

of containers, as well as the ecosystems of repositories and orchestration tools. Our work assumes that individual containers and repositories are secure, instead focusing on external threats to the mechanisms by which microservices interact.

Analysis of Consensus Protocols: Some of the attacks that we propose target the RAFT protocol used to form a service mesh. Some previous work, most notably by Sakic, *et al.* [32], examines the availability and response time of nodes participating in RAFT. However, previous work does not consider the influence of an adversary, and is instead concerned with the performance of RAFT in a purely-benign setting.

The considerable interest around blockchain technologies has driven the development of security studying microservice clusters specifically for running consensus protocols, such as Hyperledger Fabric [12,37]. These studies observe the threat of sybil attacks on collaborative network services, as does our work. However, blockchain technology can defeat traditional sybil attacks via proof-of-work or related protocol-level mechanisms, whereas our attacks require low-latency communication and collaboration between microservices.

Microservice Attacks: The attack vectors that we present are (to our knowledge) unreported. The actual attacks themselves, and the goals of the adversaries that we articulate in our threat model are inspired by previous work on attacks against more traditional systems. One of the most relevant studies is that of Cherny, *et al.* [4], which also proposes the use of microservice containers as a vector of attacks, thus providing a motivation for services as a target. In Csikor, *et al.* [7], the authors study how specially tailored access control policies crafted by an attacker may result in an exhaustion of cloud resources resulting in a denial-of-service to a cluster.

6 Conclusions

Due to the increase of deployed microservices, service mesh tools appear to be an enticing solution to manage and maintain these deployments. However, it is necessary to assess the available security mechanisms and their strength in deterring adversarial efforts. As the initial study of service mesh tools used for microservice deployments, we examine the three most popular, state-of-art offerings in the service mesh domain and articulate a threat model tailored to concerns within the service mesh domain.

Through experimentation, we find that under configuration by a skilled administrator, in 10 of the 20 studied scenarios, complete cluster compromise is possible for an attacker. Further, in 5 additional scenarios, at least one adversarial goal is achievable. Under default configuration, all studied tools, except Linkerdv2, fail to enable *any* of their security mechanisms. These results and our observations in usability of these mechanisms indicate important design flaws in the security of service mesh tools requiring further research and development.

Acknowledgments. The authors would like to acknowledge Seena Saiedian for their contributions in proofreading and revising this work.

References

1. HashiCorp. Download Consul. https://www.consul.io/downloads. Accessed 06 2020
2. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Microservices architecture enables DevOps: migration to a cloud-native architecture. IEEE Softw. **33**(3), 42–52 (2016)
3. Chen, L.: Microservices: architecting for continuous delivery and DovOps. In: 2018 IEEE International Conference on Software Architecture (ICSA), pp. 39–397, April 2018
4. Cherny, M., Dulce, S.: Well, that escalated quickly! how abusing docker api led to remote code execution, same origin bypass and persistence in the hypervisor via shadow containers. In: BlackHat 17 (2017)
5. Christopherson, J.: Spaceflight uses HashiCorp consul for service discovery and runtime configuration in their hub-and-spoke network architecture. https://www.hashicorp.com/blog/spaceflight-uses-hashicorp-consul-for-service-discovery-and-real-time-updates-to-their-hub-and-spoke-network-architecture/. Accessed 02 2020
6. Combe, T., Martin, A., Di Pietro, R.: To docker or not to docker: a security perspective. IEEE Cloud Comput. **3**(5), 54–62 (2016)
7. Csikor, L., Rothenberg, C., Pezaros, D.P., Schmid, S., Toka, L., Rétvári, G.: Policy injection: a cloud dataplane DoS attack. In: Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, pp. 147–149 (2018)
8. Das, A., Gupta, I., Motivala, A.: SWIM: scalable weakly-consistent infection-style process group membership protocol. In: Proceedings International Conference on Dependable Systems and Networks, pp. 303–312. IEEE (2002)
9. Fishner, K.: How BitBrains/ASP4all uses Consul for Continuous Deployment across Development, Testing, Acceptance, and Production. https://www.hashicorp.com/blog/how-bitbrains-asp4all-uses-consul/. Accessed 02 2020
10. Fishner K.: Using Consul at Bol.com, the Largest Online Retailer in the Netherlands and Belgium. https://www.hashicorp.com/blog/using-consul-at-bol-com-the-largest-online-retailer-in-the-netherlands-and-belgium/. Accessed 02 2020
11. Cloud Native Computing Foundation. CNCF Cloud Native Interactive Landscape. https://landscape.cncf.io. Accessed 01 2020
12. Gupta, D., Saia, J., Young, M.: Peace through superior puzzling: an asymmetric sybil defense. In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1083–1094. IEEE (2019)
13. HashiCorp. Consul by HashiCorp. https://www.consul.io/index.html. Accessed 01 2020
14. HashiCorp. Consul Docs. https://www.consul.io/docs. Accessed 02 2020
15. HashiCorp. Github hashicorp/consul. https://github.com/hashicorp/consul. Accessed 02 2020
16. Hashicorp. Modern Service Networking for Cloud and Microservices. https://www.hashicorp.com/resources/modern-service-networking-cloud-microservices. Accessed 01 2020
17. HashiCorp. Serf. https://www.serf.io/. Accessed 02 2020
18. HashiCorp. Vault by HashiCorp. https://www.vaultproject.io/. Accessed 02 2020
19. Buoyant Inc., Linkerd. https://linkerd.io. Accessed 01 2020
20. Docker Inc., Docker Home. https://docker.io. Accessed 02 2020
21. Istio. Github istio/istio. https://github.com/istio/istio. Accessed 02 2020
22. Istio. Istio. https://istio.io. Accessed 01 2020

23. Istio. Istio Docs. https://istio.io/latest/docs/. Accessed 02 2020
24. Jamshidi, P., Pahl, C., Mendonça, N.C., Lewis, J., Tilkov, S.: Microservices: the journey so far and challenges ahead. IEEE Softw. **35**(3), 24–35 (2018)
25. Grant Joy. Distil Networks securely stores and manages all their secrets with Vault and Consul. https://www.hashicorp.com/blog/distil-networks-securely-stores-and-manages-all-their-secrets-with-vault-and-consul/. Accessed 02 2020
26. Kubernetes. Kubernetes - Production-Grade Container Orchestration. https://kubernetes.io/. Accessed 01 2020
27. Kubernetes. Kubernetes Pods. https://kubernetes.io/docs/concepts/workloads/pods/. Accessed 02 2020
28. Lewis, I.: What are Kubernetes Pods Anyway? https://www.ianlewis.org/en/what-are-kubernetes-pods-anyway. Accessed 02 2020
29. Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K., Zhou, Q.: A measurement study on linux container security: attacks and countermeasures. In: Proceedings of the 34th Annual Computer Security Applications Conference, pp. 418–429 (2018)
30. Martin, A., Raponi, S., Combe, T., Di Pietro, R.: Docker ecosystem-vulnerability analysis. Comput. Commun. **122**, 30–43 (2018)
31. Rastogi, V., Davidson, D., De Carli, L., Jha, S., McDaniel, P.: Cimplifier: automatically debloating containers. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 476–486 (2017)
32. Sakic, E., Kellerer, W.: Response time and availability study of RAFT consensus in distributed SDN control plane. IEEE Trans. Netw. Serv. Manage. **15**(1), 304–318 (2018)
33. Saltzer, J.H.: Protection and the control of information sharing in multics. Commun. ACM **17**(7), 388–402 (1974)
34. Shu, R., Gu, X., Enck, W.: A study of security vulnerabilities on docker hub. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 269–280. Association for Computing Machinery, New York (2017)
35. Singleton, A.: The economics of microservices. IEEE Cloud Comput. **3**(5), 16–20 (2016)
36. SSH.COM. Ssh (secure shell). https://www.ssh.com/ssh. Accessed 02 2020
37. Sukhwani, H., Martínez, J.M., Chang, X., Trivedi, K.S., Rindos, A.: Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric). In: 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS), pp. 253–255. IEEE (2017)
38. Sun, Y., Nanda, S., Jaeger, T.: Security-as-a-service for microservices-based cloud applications. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), pp. 50–57. IEEE (2015)
39. Thomson, R.: LogicMonitor uses terraform, packer & consul for disaster recovery environments. https://www.hashicorp.com/blog/logic-monitor-uses-terraform-packer-and-consul-for/. Accessed 02 2020
40. Yarygina, T., Bagge, A.H.: Overcoming security challenges in microservice architectures. In 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), pp. 11–20. IEEE (2018)