



UvA-DARE (Digital Academic Repository)

Evaluation of Container Overlays for Secure Data Sharing

Shakeri, S.; Veen, L.; Grosso, P.

DOI

[10.1109/LCNSymposium50271.2020.9363266](https://doi.org/10.1109/LCNSymposium50271.2020.9363266)

Publication date

2020

Document Version

Final published version

Published in

Proceedings, 2020 IEEE 45th Local Computer Networks Symposium on Emerging Topics in Networking

License

Article 25fa Dutch Copyright Act (<https://www.openaccess.nl/en/policies/open-access-in-dutch-copyright-law-taverne-amendment>)

[Link to publication](#)

Citation for published version (APA):

Shakeri, S., Veen, L., & Grosso, P. (2020). Evaluation of Container Overlays for Secure Data Sharing. In H.-P. Tan, L. Khoukhi, & S. Oteafy (Eds.), *Proceedings, 2020 IEEE 45th Local Computer Networks Symposium on Emerging Topics in Networking: LCN Symposium 2020 : 17-19 November 2020, Sydney, Australia* (pp. 99-108). IEEE.
<https://doi.org/10.1109/LCNSymposium50271.2020.9363266>

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (<https://dare.uva.nl>)

Evaluation of Container Overlays for Secure Data Sharing

Sara Shakeri

Multiscale Networked Systems group
University of Amsterdam
Amsterdam, The Netherlands
s.shakeri@uva.nl

Lourens Veen

Netherlands eScience Center
Amsterdam, The Netherlands
l.veen@esciencecenter.nl

Paola Grosso

Multiscale Networked Systems group
University of Amsterdam
Amsterdam, The Netherlands
p.grosso@uva.nl

Abstract—There are many organizations interested in sharing data with others, and they can do this only if a secure platform is available. Such platforms, often referred to as Digital Data Marketplaces (DDMs), require that all of the transactions follow the agreements which are established by the participating organizations. However, translating high-level sharing policies and setting up such an infrastructure is still a big challenge.

Our work shows that containers and overlay networks can be deployed to construct a sharing platform considering security and performance aspects. We introduce an architecture for handling sharing requests in a container-based platform with focusing on improving security. We define three container connectivity: Overlay per DDM, Overlay per request, and Overlay per group. Our security analysis shows that the method "Overlay per request" is more secure against cross-container attacks. In terms of the time taken to complete the sharing requests, the difference between methods is small.

I. INTRODUCTION

Data sharing is becoming increasingly important in science as well as in industry. Combining shared data allows for richer analysis and deeper insights, such as in many Machine Learning applications. This can only be achieved if this exchange fulfills the desired level of privacy and security of each participating party.

Examples with which we are familiar are in the health and logistics domains [13], [34]: in the first case, personalized medicine requires input from multiple care providers about an individual while maintaining privacy; in the second, efficient transport of goods relies on real-time data and correlation between different logistics organizations, while avoiding exposure of information that can be exploited by competitors.

Sharing data via a secure platform relies on agreed-upon sharing *policies* that determine who can access what and how. Only if the platform strictly enforces their sharing policies will organizations trust to share their data on it.

Digital Data Marketplaces (DDMs) aim to meet this demand, and significant research efforts are ongoing at the moment to develop DDM prototypes [34].

In general, the participating organizations in a DDM can share two kinds of resources: software and data [19], [33]. The major consideration in a DDM is that all of the transactions between algorithm suppliers and data suppliers and their customers have to be done based on the pre-established

policies [23], [24]. These policies should be deployed when digital resources are being shared. We call them *DDM policies* in the rest of this paper. However, architectures for secure data-sharing platforms and methods for enforcing these high-level policies in the infrastructure in practice are still a matter of research.

Our contribution to the DDM development efforts focuses on the mechanisms for DDM policy enforcement in the infrastructure. To achieve this goal, we propose to use containerization [3]. More specifically we propose to use overlay networks to provide the connection between containers and at the same time provide isolation between sharing requests by deploying the appropriate network configuration [11], [14].

Containers are a lightweight virtualization solution that shares the OS kernel. Unlike virtual machines (VMs), they have better resource utilization and are easier and faster to deploy. In a container-based data sharing platform, containers are acting on behalf of participating parties. Therefore, each sharing request consists of a number of containers (depending on the number of participating parties) that are connected together for performing sharing transactions. However, due to lack of isolation in container-based setups, in a sharing environment constructed from containers, data confidentiality is at risk. Providing more isolation in a container-based network will decrease the probability of specific kinds of attacks and this will improve network security [31]. There are two types of isolation that should be provided in a container-based network: 1) isolation between containers and their host and 2) isolation between containers themselves. Multiple studies have been done with the focus of bringing isolation between containers and their host suggesting hardware and software solutions by utilizing Linux kernel security modules [9], [17], [25]. However, there lacks an in-depth study of providing isolation between containers themselves, which is of prime importance for improving security especially in a data sharing platform. In fact, lack of isolation between containers of sharing requests may lead to different kinds of attacks between containers like ARP spoofing or MAC flooding that will affect the shared data confidentiality [22], [26]. In this paper, we define different possible container connectivity types in a container-based DDM with the goal of improving security by controlling cross-

container connectivity and providing isolation between sharing requests.

To this end, by utilizing the proposed architecture, we implement three different overlay setup methods according to container connectivity types and study how they provide isolation and consequently security between containers of sharing requests. We also take the performance of each overlay setup into consideration by measuring the required time to complete a sharing request in each method. Trade-offs between security and performance mean that selecting the best overlay setup method highly depends on specific requirements in each DDM and their relative importance.

More specifically, the main contributions of this paper are:

- We present a container-based architecture for data sharing infrastructure that can translate high-level DDM policies to respective network configurations and run data sharing requests in practice.
- We present three methods for bringing container connectivity in the proposed architecture with the goal of improving security with higher isolation between containers of sharing requests. The presented methods are implemented by container overlays and are called Overlay per DDM, Overlay per request, and Overlay per group.
- We study the security aspects of the proposed methods with respect to how they are secure against the cross-container type of attacks. In addition, we present a performance evaluation regarding the time taken to complete a sharing request.

II. DDM POLICIES

The most important part of a secure data sharing platform is the collection of DDM policies that set the permission and prohibition rules related to a specific object in a specific location. In a data sharing platform, all of the sharing transactions have to adhere to these established policies to enable secure digital collaboration. Therefore, in a secure DDM, automatic handling of users' requests based on sharing policies has to be supported. This requires a general description model for DDM policies that leads to a more straight-forward request handling in infrastructure. We used and extended the Open Digital Rights Language (ODRL) to describe these sharing rules between participating organizations as presented in [29].

Fig. 1 shows examples of policies that can be defined in a DDM. In a specific scenario the transmission of a digital object with specific functionality (software or data) to a specific location is determined, and corresponding policies can be formulated permitting exactly this scenario. Type A policies describe processing a data set using software with two parties involved. The policies differ with respect to which organization supplies the data, which organization supplies the software, and which organization controls execution. For example, in the "Software as a Service" scenario, organization A supplies data to organization B, which processes it using its own software, and sends the result back to organization A. The corresponding policy would then authorize exactly those communications,

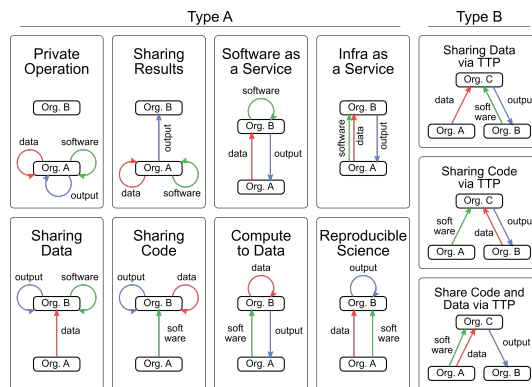


Fig. 1: Examples of DDM policies in two Types A: with two parties involved and B: with two parties and a TTP involved

and no others. Note that the "Private Operation" policy is a degenerate case, in which no exchange takes place or is permitted.

Type B policies involve two parties supplying data and software to be combined, but now there is a trusted third party (TTP) controlling execution. Corresponding policies may be constructed in a similar fashion as for Type A.

While DDM policies describe agreements between parties, each party in a DDM can define a desired service from the DDM as a *sharing request*. A sharing request can also be defined in the same format of DDM policies in terms of participating parties and the requested flow of software or input. The request can be raised by one user of a participating organization in a DDM, requesting to transfer data or software for producing and using an output.

III. CONTAINER-BASED DDM ARCHITECTURE

Fig. 2 shows the proposed architecture for constructing a container-based DDM in which DDM policies will be translated to deployable network configurations and running sharing requests will be feasible by creating requests' containers and setting up the connection between them by means of overlay setup. The main building blocks of the container-based DDM architecture are:

DDM Policy matching module: A sharing request needs to be matched with one of the pre-established DDM policies that are described by ODRL in a DDM. In the policy matching module, when a sharing request comes in, the module searches for a policy that is matched with the requested sharing scenario. If a match is found, the request will be authorized to be executed on the infrastructure and sent to the request handler. Otherwise, the request will be rejected at this level.

Request Handler: Request handler is responsible for orchestrating all of the required steps for running a sharing request: 1) Creating the list of containers that should be created for running the sharing request (Container setup) 2) Translating DDM policies to network configuration (Network policy setup) 3) Selecting the proper overlay setup for connecting the containers together (Overlay setup) and 4) Managing the implementation

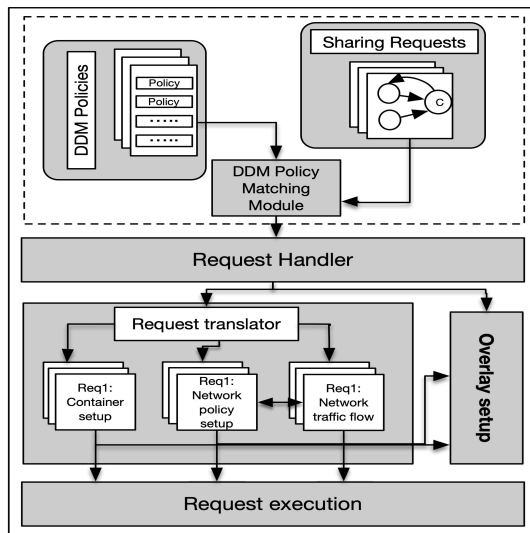


Fig. 2: Container-based DDM architecture

of the container and policy network configuration on already setup overlays (Request execution).

Overlay setup: In the proposed container-based DDM, overlays provide the connection between containers. In fact, different types of connectivity between containers can be implemented by means of overlays that lead to different levels of isolation between sharing requests. In this work we propose three overlay setups in a DDM that are described in section V.

Request translator: For implementing a DDM in practice, the high-level described DDM policies and sharing requests should be translated to the network configurations. Therefore, after receiving a sharing request, its corresponding *containers*, *network policies*, and *traffic flow* will be generated in the Request translator. Fig. 3 shows an example of the traffic flow of a sharing request between three organizations A, B, and C.

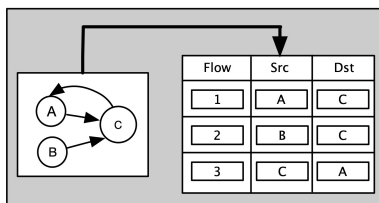


Fig. 3: A sharing request's traffic flow

Request execution: After an overlay is set up, containers and their corresponding policies are created on the overlay, and the request is executed by sending traffic between containers based on the requested traffic flow. Considering the request shown in Fig.3, three containers will be created for this request, the network policies will be deployed to allow required data transfer and then data will be transferred between these containers according to steps of traffic flow.

IV. CONTAINER CONNECTIVITY TYPES

With the goal of improving security in a DDM by isolating sharing requests' containers, different possible container con-

nectivity types in a DDM should be investigated. As a matter of fact, with less connectivity comes a higher level of isolation and consequently, the network will be more secure [31].

In this section, we define three different container connectivity types in a DDM based on the network accessibility of a container to other containers. We do this by considering that a container's network accessibility depends on overlay network configuration and the method of allocating containers to the overlay.

DDM connectivity (Fig. 4a): In this type of connectivity, all of the containers are allocated to one overlay network and therefore all of them are connected together. From a security perspective, in this type, if for example a container of a request of organization B is compromised (the white circle in the center) all of the other containers in the DDM are at risk. This type has the highest attack surface and the lowest level of isolation between requests' containers. However, this method has low overhead in terms of network setup. For setting up a DDM in this method just one overlay needs to be set up and then the network will be ready to start sharing transactions.

Request connectivity (Fig. 4b): In this type, there is one overlay for each single sharing request. Therefore, only the containers related to the same request are in the same overlay. As is shown in Fig. 4b, the connection between containers is automatically limited by just being in the same overlay. In this connectivity type, if a container is compromised, only the containers that are related to that sharing request will be affected. Therefore, this method has the highest level of isolation.

However, in this type, one overlay has to be set up for each single sharing request to connect its containers together. This will lead to a substantial delay in setting up the network and will negatively affect the time of completing a sharing request. By increasing the number of sharing requests, this delay can disrupt the network availability that in delay-sensitive requests is not negligible.

Group connectivity (Fig. 4c): As an intermediate type between the two previous ones, we define Group connectivity type. In this type, the requests and their respective containers will be assigned to different groups. Containers related to the same group will be in the same overlay network. For grouping the requests, we consider two characteristics:

- The set of participating organizations in the DDM: this is the list of organizations in a DDM that are involved in sharing transactions in a DDM. For example in Fig. 1, there will be two lists of participating organizations. (A, B) for sharing requests of *Type A* and (A, B, C) for sharing requests of *Type B*.
- The owner of the request: this is the organization that has submitted the request and will use the output of the request. Referring back to the "Sharing Results" scenario in Fig. 1, organization A is the owner of the request because it is using the output of the sharing transactions.

All sharing requests can be expressed with the same formalism, as Group ((Set of participating parties), owner of request).

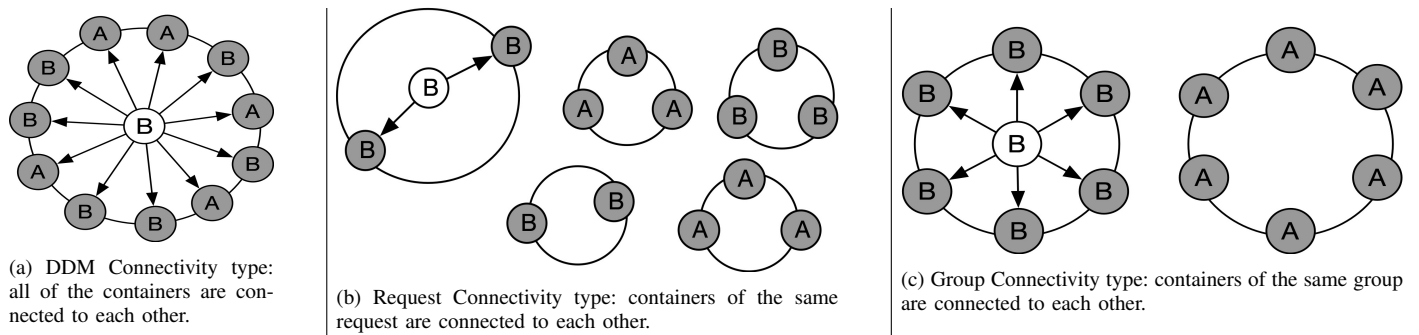


Fig. 4: Three container connectivity types in a DDM

For example Group ((A,B),A) defines Two organizations A, B are involved in sharing transactions and requests have been submitted by organization A. Therefore, in a DDM with two organizations A, B, and a TTP (organization C), four different groups are defined. As is shown in Fig. 4c, organization A requests' containers are isolated from organization B requests' containers. In this case, if a container is compromised (the white circle) only the containers in the same group will be affected, which means that the attack surface is less than the DDM connectivity type. In addition, the number of overlay networks that have to be set up matches the number of groups. This leads to less delay in network setup time compared to the Request connectivity type.

V. OVERLAY SETUP

Overlay setup plays an important role in providing this isolation. In this section, we present the method of setting up DDMs according to container connectivity types by means of container overlays and explain the policy enforcement method in each setup. In the following we focus on DDMs with three participating organizations, but that this is just to show the overall operations of the system and that the insights are clearly applicable. In all setups, for constructing a DDM with organizations A, B, and C, we consider three machines acting as the organization's node in the DDM. Containers of each organization will be created on its own node. Note that regardless of which organization has submitted the request, containers act on behalf of participating organizations and will be created on the organizations' node.

1) **Method 1: Overlay per DDM (Fig. 5a):** This method constructs the DDM according to the *DDM connectivity* type. In this method, all of the containers related to different sharing requests will be running inside one overlay network for the whole DDM. For setting up this configuration, we created one Kubernetes [7] cluster and connected all of the containers in the cluster by a Calico overlay network [1]. We selected Calico as it is deployable in most cloud environments in addition to being an efficient and scalable container networking plugin that is integrated with Kubernetes. Calico uses BGP for routing among worker nodes. A Calico node contains two processes: Felix and Bird. Felix programs host route tables and Bird is responsible for route sharing among nodes [10]. After installing Calico, it

uses IP-in-IP for encapsulating container's packets, which are then routed by the host through a specific interface. With this implementation all containers are connected to each other at layer 3 and inside one overlay network.

Policy enforcement method: In this method, DDM policies are enforced by Calico network policy rules. Calico filters the traffic between containers by generating *iptables* rules in the host machine of containers. Considering data flow in Fig. 3 as an example, the policy rules of any request allows any traffic according to request traffic flow (source and destination) and forbids any traffic from any other container.

```

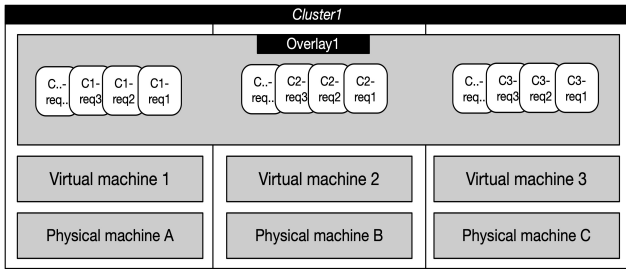
1 kind: GlobalNetworkPolicy
2 metadata:
3   name: 'req1-from-org.A-to-org.B'
4 spec:
5   selector: id.container == 'org.A'
6   types:
7     - Egress
8   egress:
9     - action: Allow
10      destination:
11        selector: id.pod == 'org.B'
12 ---
13 kind: GlobalNetworkPolicy
14 metadata:
15   name: 'req1-to-org.B-from-org.A'
16 spec:
17   selector: id.container == 'org.B'
18   types:
19     - Ingress
20   ingress:
21     - action: Allow
22      source:
23        selector: id.pod == 'org.A'
24

```

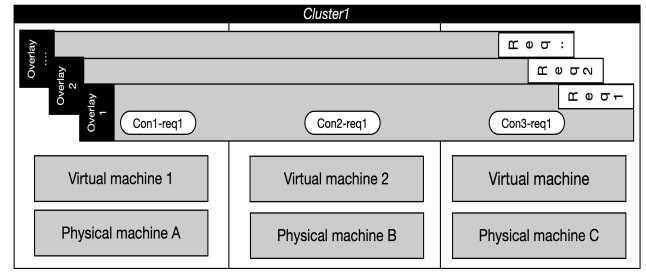
Listing 1: An example of Calico network policy that allows traffic from container org.A to container org.B related to request1.

Listing 1 shows an example of Calico network policy deployed for "Sharing Results" scenario in Fig. 1. It allows any traffic from a container of organization A to a container of organization B and forbids any other traffic to these two containers.

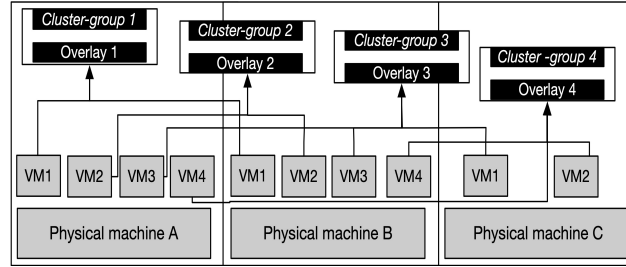
2) **Method 2: Overlay per Request (Fig. 5b):** In this method, all of the connections are based on the *Request connectivity type*. We used Docker Swarm cluster technology for implementing this method as it is the available technology for running



(a) Overlay per DDM network setup, one cluster consisting of three virtual machine



(b) Overlay per Request network setup, one cluster consisting of three virtual machines



(c) Overlay per Group network setup, four clusters consisting of one or two virtual machines

Fig. 5: Overlay network setup in a DDM

multiple overlays between the same nodes in one cluster [12], [18]. We first create a specific overlay for each request and then create related containers inside that overlay network. Therefore, there is one specific bridge for each requests' container in each host, and containers of different requests are not connected to each other. Fig. 5b shows that as an example four different overlay networks have been created for running four sharing requests in DDM. Docker Swarm uses VXLAN for packet encapsulation in overlay.

Policy enforcement method: DDM policies are enforced by separating sharing requests via one overlay per request. In fact, defining firewalling rules between containers is not possible and the connection should be confined by overlays. Unlike in the overlay per DDM method, where containers are connected to each other in the network layer but traffic is controlled by filtering rules, in this method there is no network connectivity between containers of two different requests.

3) **Method 3: Overlay per Group (Fig. 5c):** This method implements the *Group connectivity type*. A separate overlay is created for each group and the traffic of requests' containers of each group should be filtered by network filtering rules. We could not use Swarm as it can not implement the filtering rules between containers of a group. Therefore, we used Kubernetes. Considering the four groups that are needed for constructing a DDM with two participating organizations and one other organization as a TTP, we need four overlay networks. As we can just create one overlay in each Kubernetes cluster, we created four Kubernetes clusters, one for each group. Depending on the participating organizations, two or three virtual machines are involved in each cluster. We used Calico as overlay technology in each cluster. As a result, all of the containers related to the

same group are connected via one overlay network, but there is no connection between the containers in different groups.

Policy enforcement method: Policy enforcement in this method is implemented using Calico network policies. We define the Calico network policy based on DDM policies and the permitted traffic flow of a request for containers inside a group.

VI. SECURITY

In a sharing transaction three main aspects of security should be provided in a DDM.

Availability: All of the resources should be available for running authorized sharing transactions by organizations. In other words, during running multiple sharing requests at the same or different hosts, all of the related containers should be available for any kind of legitimate traffic transmission.

Confidentiality: Access to unauthorized data should be forbidden. When a container is compromised it may be able to have access to unauthorized data related to another request that is stored in the host machine.

Integrity: An organization that is not authorized should not be able to change data. This can be violated when a compromised container can have access to data related to other organizations and stored in the host machine.

[31] provides a clear classification of kinds of attacks that can happen in a container-based platform: application to container, host to container, container to host, and container to container. Given our focus in this research, we consider only the container-to-container attack type; the other three types of attacks are out of scope of this paper.

In *Container to container* kind of attacks a compromised container executing a request in a DDM attacks containers of

TABLE I: Cross-container attack analysis of method 1(Overlay per DDM), method 2(Overlay per request), and method 3(Overlay per group)

Attack Type	Attack Scenario	Security		
		Method1	Method2	Method3
ARP Spoofing	The compromised container can have access to all unauthorized data sets which are related to other requests	High	High	High
Malware Spread	A malicious container may spread a malware across container that is connected to	Low	High	Medium
L3 DoS Attack	A compromised container might flood the other container related to another request at layer 3 by ping of death or ICMP flooding	Low	High	Medium
Application DoS Attack	A compromised container might flood the other container related to another request at application layer by SYN flood or HTTP flood	High	High	High

other sharing requests. Other containers can be in the same or in different hosts. We classify the container to container attacks that are presented in [31] to three main possible attack scenarios:

ARP Spoofing: A compromised request’s container may gain access to confidential data via a ARP (Address Resolution Protocol) spoofing attack on other requests’ containers. Method 1 is secure against this kind of attack because Calico makes the connection between containers on layer 3 and therefore, ARP spoofing can not happen [16]. This is also true about method 3, as it also uses Calico. In method 2 as the containers are connected to different bridges and they are not in the same local area network, ARP spoofing can not happen between them. Therefore, these three methods are all secure against ARP Spoofing.

Malware Spread: A malicious container may spread malware across multiple containers that are connected to it. This is an east-west traffic that may not be detected by network policies because there is no detection of the content of transferred data among containers. In this type of attack, all of the containers that are connected to the malicious container are at risk and the *confidentiality* and even *integrity* of DDM is affected. Comparing the three different methods, as there is no network connection between each two request’s containers in method 2, it can provide more security than method 1. As in method 3 containers are distributed in groups, its security against this kind of attack is higher than method 1 and lower than method 2.

Denial of Service (DoS) Attacks: In DoS attacks in a container-based infrastructure, a compromised container can send a huge amount of traffic to other containers, interrupt their service, and affect the *availability* of DDM. We classify possible DoS attacks into two categories of "L3 DoS attacks" and "Application layer DoS attacks".

In L3 DoS attacks, for example, a malicious container overwhelms the other container by sending a large number of echo-requests to affect its functionality. The isolation level between containers of different requests plays a major role in mitigating this kind of attack. As there is no network connection between different requests’ containers in method 2, it is the most secure method comparing to two other methods. However, in method 1 containers are all connected together and it does not have a mechanism for mitigating this kind of attack. Method 3 is more secure than method 1. It takes advantage of the distribution of requests between different overlay groups and

this will decrease the number of requests that may be affected by this kind of attack.

In application layer DoS attacks, due to the fact that in the current setup of all methods, no session can be established between containers of different requests all of the methods are secure.

Table I summarizes the security analysis of proposed methods. It shows the degree of security of each method. We defined high, medium and low as qualitative metrics where high means more protection against the attack considered. Method 2 provides the most security against all kinds of attacks, because setting up an overlay network for every single request increases cross-container isolation between different requests’ containers.

VII. PERFORMANCE ANALYSIS

In this section we analyze the performance of the proposed methods by measuring the time taken to complete a sharing request in each method.

A. Experiment settings

Hardware specification: Our experiments were performed on three servers, connected by 10 Gigabit Ethernet. Each server is equipped with a dual 10-core Intel Xeon E5-2690 2.9GHz processor and 8GB memory.

Software specification: We used Ubuntu 18.04 and Linux kernel 4.15.0 as the host OS, Docker Community Edition 18.09, Kubernetes 1.18 for managing containers, and Calico version 3.8 to implement the overlay networks.

We performed a number of experiments aimed at assessing the times required to complete a request in the proposed methods. In each experiment, we first selected the type of requests, either Type A or Type B based on the pattern shown in Fig. 1. We then, simultaneously, submitted a group of requests consisting of a mix of sharing scenarios from the chosen type. Following group sizes are considered:

$$group_size \in \{10, 20, 30, 40, 50\}$$

Each sharing request consists of two main steps: 1) setting up the network, and 2) transferring the shared data based on the request’s traffic flow. We measured the average network setup time, the average transfer time, and finally the average total time for completing a request. We repeated every experiment three times to ensure consistency of the results. For every repeat, the mean of the quantity of interest was calculated across the group of requests. In the plots the mean and standard deviation of these means is shown.

Setting up the network in methods 1 and 3 involves creating the request's containers and deploying the network policies between them. In method 2, an overlay is established for each request and the request's containers are then allocated to the overlay. Next, the shared data transfer is executed by sending 1 GByte traffic for each traffic flow of the request using *iperf3* [5].

B. Experiment results

- **Setup time:** Fig. 6 shows the average setup time of each request. Setup time increases in all three methods with increasing number of requests. Method 2 has the largest setup time and method 3 has the smallest. Note that in our experimental setup, in method 3 two clusters are used and half of the group size is running in each cluster. Therefore, setup time of method 3 should be compared to the setup time of method 1 at half the group size. For example, setup time of 40 requests in method 3 is almost equal to setup time of 20 requests of method 1.

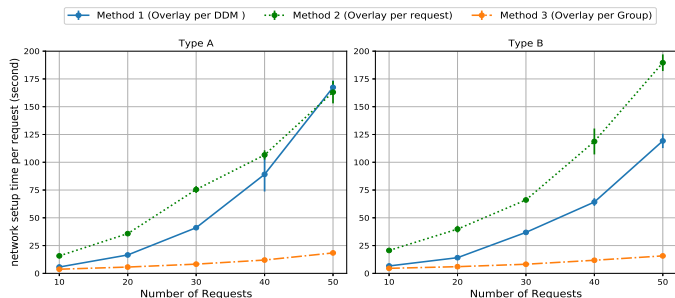


Fig. 6: Network setup time as a function of the number of requests for the three methods. Setting up an overlay network (green) takes more time than configuring traffic filters (blue) within an existing overlay network. Using one overlay per group is fastest, but most of the apparent difference is due to having more resources available (see main text).

- **Transfer time:** Fig. 7 shows the average transfer time. Transfer time in Type B is larger than Type A for all methods. That is because of the difference between the number of transactions in requests of Type A (1.43 on average) and requests of Type B (3). As for the setup time, method 3 is faster than method 1, however, it has more resources available.

These performance results differ between requests of Type A and Type B. Also, for Type A and method 2, transfer time decreases with increasing group size, which is unexpected. For Type B, a decrease relative to method 1 is also visible for larger group sizes.

To investigate this further we plot the exact transfer time of each individual request for different group sizes for both type A and type B of Method 2 (Fig. 9). When increasing the number of requests and having more containers to set up at the same time, the average setup time increases, as was shown in Fig. 6. As the group size becomes larger more requests are running in parallel, which increases resource contention and slows down the requests.

For Type A (Fig. 9, top row), less data is transferred and the transfer step is shorter. As result the cluster spends more time in setting up and shutting down the networks. Therefore, the requests are scattered in time, which leads to fewer transfers running in parallel and lower average transfer time. This effect becomes larger for larger group sizes and it explains the decrease of transfer time for Type A groups over 30.

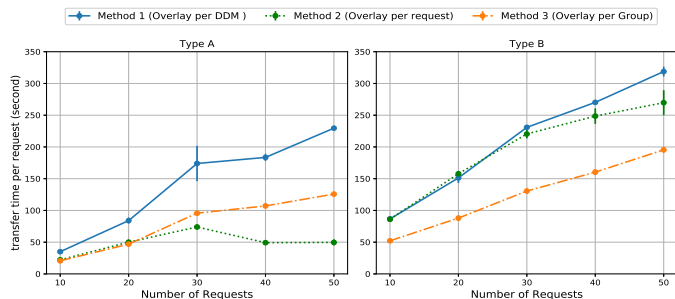


Fig. 7: Transfer time as a function of the number of requests in three methods. A separate overlay network per request appears to be much faster than using a single network, but is mostly a result of the requests being scheduled differently. See Fig. 9 and the text.

- **Total time:** Fig. 8 shows the average total time of completing a request. The overall time for Type B is more than Type A in all methods. For Type A, average total time in method 2 is less than method 1, whereas for Type B method 1 is faster. In both types of requests method 3 is taking less time, and roughly half of the time of method 1, that is due to the fact that it has more resources.

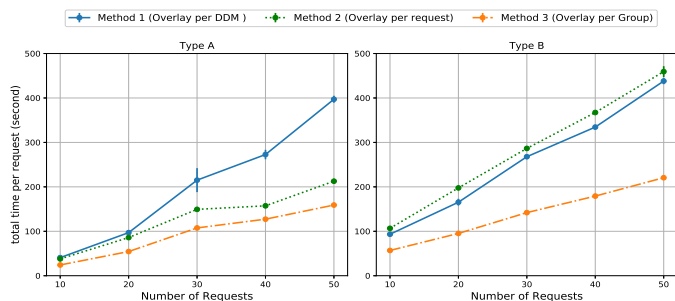


Fig. 8: Total time to complete a request as a function of the number of requests in three methods. For Type A requests, an overlay per request was measured to be faster, due mainly to the way the requests were scheduled here. The results for Type B are more representative, and show that an overlay per request is slower than using a single overlay network for the whole DDM. Overlay per group is on par with overlay per DDM when resource differences are taken into account.

VIII. DISCUSSION

In this section we will discuss in more detail the implications of our findings. We are particularly interested in how the three

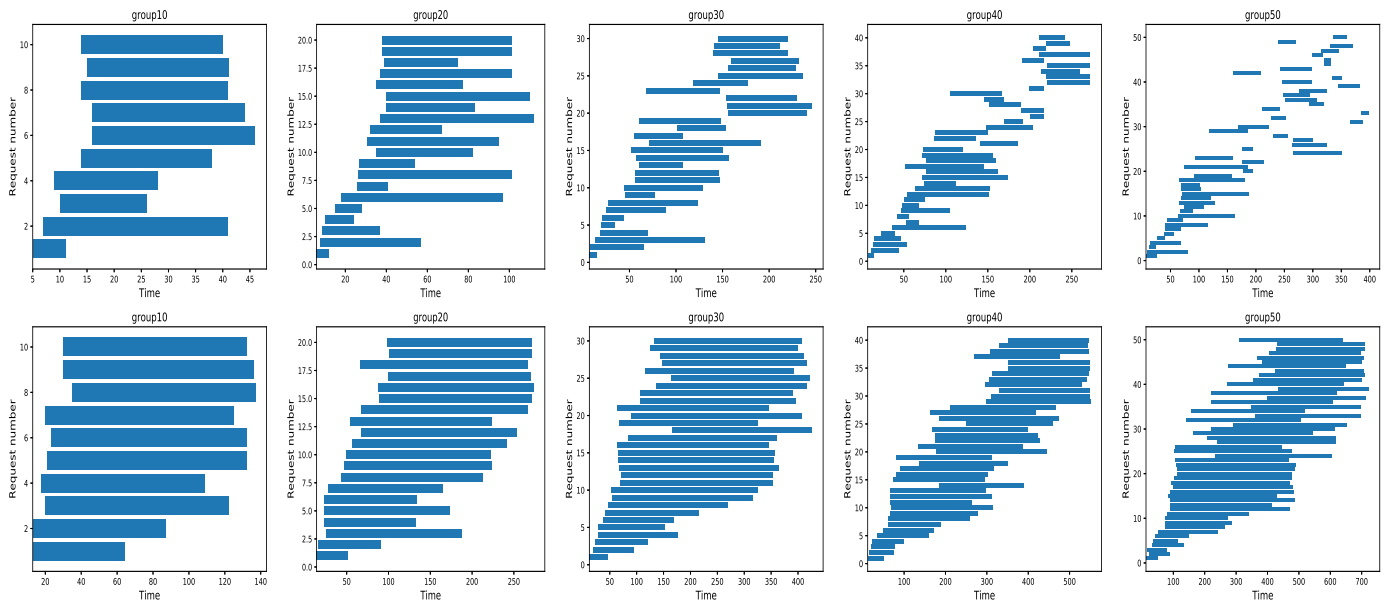


Fig. 9: Transfer time of each request for Method 2 and different group sizes, for Type A (top row) and Type B (bottom row). The requests are submitted at time zero. Each request's bar starts when the setup step of the request is done and the transfer step starts, and ends when the transfer finishes. For large group sizes (rightmost plots), there is on average less overlap between the requests, causing them to complete more quickly. The total time required to execute all requests is still larger for larger groups of requests. The effect is much stronger for Type A (top row) than for Type B (bottom row) requests.

methods compare with respect to security and performance, so an optimal setup can be chosen when implementing a DDM. For Type A, in our experiment method 2 was faster than the other methods. However, our results indicate that this may be partially caused by the Swarm scheduler, and more research is needed into the scheduling behaviour of Kubernetes and Swarm to see how this affects performance for this use case. For Type B, method 2 is slightly slower than the others, and this can be considered as a more general result.

While method 2 is the slowest method, it is also the most secure one. Method 1 is the fastest but the least secure, and method 3 is in between from both perspectives. In general, the performance difference between methods is small however, so in most cases method 2 will be preferred.

The presented experiments show the performance of the proposed methods when the system is under pressure. In a real-world system, loads will vary with time. In these experiments, all of the requests arrived at the same time, which can be considered as a worst-case scenario. However, the results are consistent across different load levels which suggest that the conclusions will hold for lower load levels as well.

IX. RELATED WORK

We expect the adoption of DDMs to increase in the coming years. The research and efforts to arrive at working platforms are ongoing. Several initiatives tackle the problem to establish the contracts between parties. For example in the Dutch logistics sector, which we are familiar with, has defined iSHARE [6]. iSHARE is a uniform set of agreements for identification,

authentication, and authorization to share the logistics data in a safe and controlled fashion. This system can be used by all parties which have activity in the logistics sector. However, efforts like this do not define, as we do, an effective architecture for deploying the contracts and agreements in infrastructure to make a DDM work in practice, by using container overlay networks.

In regard to the evaluation of overlay technologies' performance there is a number of previous studies that have provided us with guidelines. The authors in [21] investigated the possibility of deploying Osmotic Computing environments in order to deploy distributed microservices among Cloud, Edge, and IoT devices. In particular, they deployed two different microservices: FTP and CoAP inside Docker containers orchestrating by Kubernetes. In order to find the best overlay solution, they performed scalability analysis on four different network overlays: OVN [15], Calico, Weave [8], and Flannel [4]. [27] proposes a solution for connecting containers utilizing EVPN and ILA as overlay technologies. They study the performance of Cilium/eBPF in network filtering. Authors in [30], evaluate the scalability of Calico and Cilium [2] as two popular overlay technologies by measuring the network throughput with increasing the number of containers and the number of deployed network policies between containers. Finally, the work in [32] presents a performance analysis of different methods of bringing the network connectivity between containers including overlays. In our work, we do consider these efforts and we move further to identify the better-suited overlay setups depending in relation to the data sharing request characteristics.

Different methods have been studied for providing security in docker containers. For example, [26] and [20] utilize Linux Kernel security modules like Apparmor [9] and SELinux [17] to enhance access control mechanism of the containers and provide more protection of the host against a malicious container. [25] studies a virtualized trusted platform (vTPM) in a container-based architecture for protecting containers from a malicious host. The main focus of these works is about limiting the container's ability to access the resources of the host but not about the connection between containers in the network layer.

Authors in [22] discuss important security issues of Docker containers and proposed solutions. They also propose an algorithm to tackle Dos attack issues by limiting container resources. [28] perform a comprehensive study about security of docker containers and denotes the possible vulnerabilities in docker containers and the available solutions in literature works. It also specifically investigate the inter-container attacks and suggest container network separation method as a solution, however, no practical solution is presented. In this paper, we focus on providing the network layer isolation between containers by means of overlay setups specifically for data sharing services in a DDM.

X. CONCLUSION

We propose in this paper a container-based Digital Data Marketplace, and we introduce an architecture for implementing sharing requests and deploying high-level DDM policies in infrastructure. We studied the use of container overlay networks for this application and how they affect the security and performance of the network.

We propose three different methods for setting up overlay networks between containers which provide different levels of isolation. To evaluate each method, we study how they are secure against cross-container network attacks. This work shows that the 'Overlay per request' method is more secure than the other methods as it provides better isolation between requests. We also compared the time required to complete a sharing request between the methods. The three methods perform similarly, although the "Overlay per request" method is slower than the others in larger type of requests.

Our future work will focus on building a more complete DDM involving multiple sites and applications running across the DDM using a cross-domain overlay network. We also want to evaluate if a per-request selection of overlay setup methods can be more efficient while providing sufficient security. Finally, we intend to investigate how dynamic programmable network architectures can be used to improve system performance and security.

ACKNOWLEDGMENT

This work is supported by the Netherlands eScience Center and NWO under the project SecConNet. We want to specifically thank Rena Bakhshi for the useful discussions and feedback.

REFERENCES

- [1] "Calico," <https://docs.projectcalico.org/v2.0/introduction/>, 2019, [Online; accessed Jan-2020].
- [2] "Cilium," <https://docs.cilium.io/en/v1.5/>, 2019, [Online; accessed Jan-2020].
- [3] "Docker," <https://www.docker.com/>, 2019, [Online; accessed Jan-2020].
- [4] "Flannel," <https://github.com/coreos/flannel/flannel>, 2019, [Online; accessed June-2019].
- [5] "iPerf," <https://iperf.fr/>, 2019, [Online; accessed Jan-2020].
- [6] "iSHARE," <https://www.ishareworks.org/en/node/6>, 2019, [Online; accessed June-2019].
- [7] "Kubernetes," <https://kubernetes.io/docs/tutorials/kubernetes-basics/>, 2019, [Online; accessed Jan-2020].
- [8] "Weave Net," <https://www.weave.works/oss/net/>, 2019, [Online; accessed Jan-2020].
- [9] "AppArmor," <https://wiki.archlinux.org/index.php/AppArmor>, 2020, [Online; accessed May-2020].
- [10] "Calico Routing Modes," <https://octez.com/docs/2020/2020-10-01-calico-routing-modes/>, 2020, [Online; accessed May-2020].
- [11] "Container Networking," <https://thenewstack.io/container-networking-breakdown-explanation-analysis/>, 2020, [Online; accessed Jan-2020].
- [12] "Docker overlay networks," <https://docs.docker.com/network/overlay/>, 2020, [Online; accessed May-2020].
- [13] "Enabling Personal Intervention," <https://delaat.net/epi/>, 2020, [Online; accessed Jan-2020].
- [14] "Networking," <https://docs.docker.com/v17.09/engine/userguide/networking/>, 2020, [Online; accessed Jan-2020].
- [15] "OVN," <https://github.com/ovn-org/ovn-kubernetes>, 2020, [Online; accessed Jan-2020].
- [16] "Prevent DNS (and other) spoofing with Calico," <https://www.tigera.io/blog/prevent-dns-and-other-spoofing-with-calico/>, 2020, [Online; accessed May-2020].
- [17] "SecurityEnhancedLinux," <https://en.wikipedia.org/wiki/Security-Enhanced-Linux>, 2020, [Online; accessed May-2020].
- [18] "Use bridge network," <https://docs.docker.com/network/bridge/>, 2020, [Online; accessed May-2020].
- [19] M. Ali, R. Dhamotharan, E. Khan, S. U. Khan, A. V. Vasilakos, K. Li, and A. Y. Zomaya, "Sedasc: Secure data sharing in clouds," *IEEE Systems Journal*, vol. 11, no. 2, pp. 395–404, June 2017.
- [20] E. Bacis, S. Mutti, S. Capelli, and S. Paraboschi, "Dockerpolicymodules: Mandatory access control for docker containers," in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 749–750.
- [21] A. Buzachis, A. Galletta, L. Carnevale, A. Celesti, M. Fazio, and M. Villari, "Towards osmotic computing: Analyzing overlay network solutions to optimize the deployment of container-based microservices in fog, edge and iot environments," in *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, May 2018, pp. 1–10.
- [22] J. Chelladhurai, P. R. Chelliah, and S. A. Kumar, "Securing docker containers from denial of service (dos) attacks," in *2016 IEEE International Conference on Services Computing (SCC)*, 2016, pp. 856–859.
- [23] L. Gommans, J. Vollbrecht, B. G. de Bruijn, and C. de Laat, "The service provider group framework: A framework for arranging trust and power to facilitate authorization of network services," *Future Generation Computer Systems*, vol. 45, pp. 176 – 192, 2015.
- [24] D. Harris, L. Khan, R. Paul, and B. Thuraisingham, "Standards for secure data sharing across organizations," *Comput. Stand. Interfaces*, vol. 29, no. 1, pp. 86–96, Jan. 2007.
- [25] S. Hosseinzadeh, S. Laurén, and V. Leppänen, "Security in container-based virtualization through vtpm," in *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, 2016, pp. 214–219.
- [26] F. Loukidis-Andreou, I. Giannakopoulos, K. Doka, and N. Koziris, "Docker-sec: A fully automated container security enhancement mechanism," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1561–1564.
- [27] L. Makowski and P. Grosso, "Evaluation of virtualization and traffic filtering methods for container networks," *Future Generation Computer Systems*, vol. 93, pp. 345 – 357, 2019.
- [28] A. Martin, S. Raponi, T. Combe, and R. D. Pietro, "Docker ecosystem – vulnerability analysis," *Computer Communications*, vol. 122, pp. 30 – 43, 2018. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0140366417300956>

- [29] S. Shakeri, V. Maccatrozzo, L. Veen, R. Bakhshi, L. Gommans, C. de Laat, and P. Grosso, "Modeling and matching digital data marketplace policies," in *2019 15th International Conference on eScience (eScience)*, 2019, pp. 570–577.
- [30] S. Shakeri, N. van Noort, and P. Grosso, "Scalability of container overlays for policy enforcement in digital marketplaces," in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, 2019, pp. 1–4.
- [31] 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.
- [32] K. Suo, Y. Zhao, W. Chen, and J. Rao, "An analysis and empirical study of container networks," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 189–197.
- [33] D. Thilakanathan, S. Chen, S. Nepal, R. Calvo, and L. Alem, "A platform for secure monitoring and sharing of generic health data in the cloud," *Future Generation Computer Systems*, vol. 35, pp. 102 – 113, 2014, special Section: Integration of Cloud Computing and Body Sensor Networks; Guest Editors: Giancarlo Fortino and Mukaddim Pathan.
- [34] L. Zhang, R. Cushing, L. Gommans, C. De Laat, and P. Grosso, "Modeling of collaboration archetypes in digital market places," *IEEE Access*, vol. 7, pp. 102 689–102 700, 2019.