University of Amsterdam

Research Project 1

Zero Trust Network Security Model in containerized environments

February 9 2020

Students:
Catherine de Weever
cweever@os3.nl

Marios Andreou
mandreou@os3.nl

Abstract

In security, it is important for an organization to prevent leak of confidential data by lowering the risks of cyber-attacks. Zero Trust is a security model that treats all network traffic as hostile, even if it is inside the perimeter. However, securing workloads particularly those who are running in the public cloud is essential: Cloud assets, containers, functions and Virtual Machines (VMs) are vulnerable and are attractive targets for malicious actors.
In this research, we focus on the investigation and implementation of the appropriate operational controls, which are illustrated in figure 11, in order to implement Zero Trust to mitigate data leakage for "east/west" traffic in a containerized environment. In order to mitigate data leakage, we need to be able to regulate the traffic so we can trace and find where, how and when the attack occurs. We also need to protect the data when it is in transit to prevent man in the middle attacks. In order to implement this, we search for the necessary tools, which are going to provide regulation in depth and protection for the data that are being transferred. We demonstrate our solution by implementing a Proof of Concept with the necessary tools.
Our results show that having the appropriate tools, which provide the necessary operational controls we can apply protection to the data that are being transferred between microservices ("east-west" traffic), as well as regulating the traffic flowing between them.

1 Introduction

Not too long ago the workplace was like a castle: Data was stored and managed in one place and one needed to be physically there to access this data. Security was applied by monitoring and protecting everything that was going into or out of the premises. The basic assumption was that everything within the security perimeter can be trusted. Nowadays one do not necessarily have to set foot in the office building to come to work. Companies enable employees to access companies' assets through mobile devices and cloud software, regardless of where they're located. Valuable business data transfer continuously between SaaS applications, IaaS applications, data centers, remote users, IoT devices etc. This makes cybercriminals' life much easier. It causes a wider attack vector, meaning that there are more entry points to exploit. It can take months to locate a malicious attack.
In 2010, Forresters' John Kindervag introduced his Zero Trust security strategy in the United States. This entails that no device, user, workload or system should be trusted by default regardless of the location it is operating from, either inside or outside the security perimeter.[17]
Zero Trust builds on familiar concepts such as multi-factor authentication, identity access management, cloud orchestration, analytics, encryption, scoring and file system permissions. Zero Trust is as much about one's security approach as it is about specific tools and implementation strategies. On2IT's Zero Trust framework[18] consists of operational controls shown in figure 11 that need to be applied to implement Zero Trust.
In cloud environment, there is not only traffic travelling from service to user (North-South), but also service-to-service (East-West). "East-West" traffic can also contain confidential data that one have to protect during transition in order to prevent man in the middle attacks, as well as observing the traffic and events happening in order to be able to trace and find when, how and who tried to act maliciously to one's services.[16] Thus, this leads us to our following research question:
"How to implement Zero Trust for "east/west" traffic between microservices in containerized environment?"
To answer this, we drafted with the following sub-questions:

- *How to regulate the "east/west" traffic flow?*
- *How to implement confidentiality for transit data?*

2 Related Work

There has been some research done on how to protect data in a cloud environment while implementing Zero Trust. Here, we discuss the different approaches that have been done.

Casimer DeCusatis et al. did research on a Zero Trust approach based on a steganographic overlay, which embeds authentication tokens in the TCP packet request and first packet authentication. This approach protects data on the transport-level. It ensures prevention of fingerprinting of key resources and data protection against reconnaissance attacks. They demonstrate the application of this approach in a enterprise-class server and cloud environment. From the demonstration they showed that DDoS, fingerprinting and reconnaissance attacks were blocked. This approach provides protection on layer 3/4, but it does not on layer 7.[6]

Fatima Hussain et al. proposed the API gateway/proxy-based approach to implement Zero Trust. This approach entails implementing a secure API service mesh. They used the tools Istio[12] and Kubernetes[14] to achieve this. These tools are used in our Proof of Concept that we'll discuss in our Background section. They also found a way to automate the linking of new APIs to already existing categories of the service mesh by using a machine learning-based intelligent association model. However, they did not implement this model in a real environment.[11]

Zirak Zaheer et al. came up with a network-independent perimeterization solution for microservices called eZTrust. eZTrust focuses on finegrained, context-rich micro-service identities. It entails tracing data of microservices made available by extended Berkely Packet Filter (eBPF)[13] and classifying packets based on micro-service contexts. They tested the feasibility of their solution by implementing a Proof of Concept prototype. They also compared the performance of their solution with other approaches and concluded that their solution performed better. This solution only focuses on the tracking of traffic between microservices. Although they mentioned in the discussion section that it is possible to add encryption, encryption is not implemented in their Proof of Concept.[20]

3 Background

To get a better understanding of how our setup works we describe in this section the appropriate tools we used and their functionalities.

Google Kubernetes Engine
Google Kubernetes Engine (GKE) is a platform for containerized applications by Google. We used GKE to deploy our demo application and used Kubernetes and Istio as orchestration planes in a collaborative manner. The microservice architecture consists of nodes where the nodes are worker machines that can be virtual or physical machines. A node can consists of one or multiple pods. "A Pod is a group of one or more application containers (such as Docker or rkt) and includes shared storage (volumes), IP address and information about how to run them." according to Kubernetes Documentation [15]. When Kubernetes is used along side Istio it provides managing of availability and resource consumption of nodes and adding of pods when needed. Meanwhile Istio adds extra containers such as a sidecar proxy for security to the pod.

Istio
Istio is an open source service mesh for networking of microservices applications. The core features of Istio are the following:

- Configure rules to control the flow of traffic and API calls between services.
- Provide security at the network and application layer by securing the communication channel, managing authentication, authorization and encryption of service communication.
- Provide access control by configuring custom policies for an application.
- Provide tracing, monitoring and logging to get visibility into the performance of the services.

We are only interested in the Security aspect of Istio, so we will focus on that. The Istio service mesh consists of two parts, the data plane and the control plane. The control plane contains the following components for security:

- Pilot: Configure and deploy policies to the Sidecar proxies.
- Mixer: Manage auditing. It provides logging and monitoring of all the traffic route in a cluster.
- Citadel: Manage keys and certificates for authorization.

These components are centrally-managed and they operate independently of the applications running within the service mesh as shown in figure 1.
The data plane contains Envoy[1] proxies to secure "east-west" traffic between containers by implementing security rules. They work directly with the applications to provide local security features such as mutual TLS and routing policies. In a Kubernetes environment, the data plane contains Envoy sidecars instead. They are added to each deployed pod. The data plane communicates with the control plane via these sidecars. By adding authorization policies to the sidecar proxies, Istio provides micro-segmentation.[19]

As mentioned before, Istio provides encryption of transit data by using mutual TLS. When a container wants to communicate with another container, the following steps are taken:

1. The sidecar proxy of a container on the client side starts a TLS handshake with the sidecar proxy of a container on the server side.
2. During the TLS handshake the sidecar proxy on the client side does a secure naming check[2].
3. After the check, they establish a mutual TLS connection and Istio sends the traffic from the client side sidecar proxy to the server side sidecar proxy.
4. After authorization, the server side sidecar proxy forwards the traffic to its targeted service.

Something to note is that the data is unencrypted between the sidecar proxy and the service inside the pod.
So Istio redirects every traffic to the sidecar proxies. This redirection can be costly when for example an IP based tooling such as iptables is used: the entire TCP/IP stack has to be traversed multiple times.[4] Another problem with Istio is that it doesn't provide any protection for the sidecar proxies themselves, so they can be compromised. However, the following tool (Cilium[3]) can be a solution for these problems and can provide us more functionalities for our .

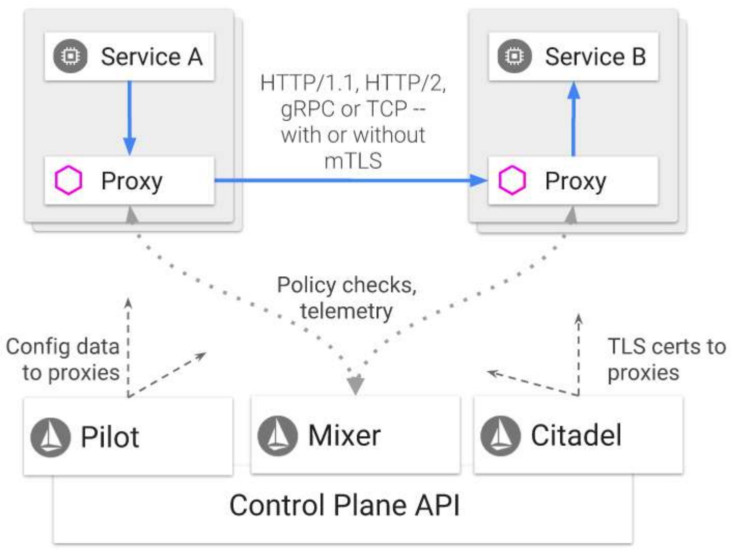


Figure 1: Istio Architecture. [5]

Cilium
Cilium is a networking plugin that can be integrated with Istio. Cilium enhances the security

features and performance of Istio. The foundation of Cilium is Berkeley Packet Filter[7], which contains one of the features called sockmap[8]. With this feature Cilium can provide filtering and redirection on a socket level and thus makes Cilium socket-aware. When this feature is applied to Istio, it accelerates the TCP connection between the application and the sidecar proxy by short-circuiting TCP sockets on the same node.

Cilium also provides L3/L4 security policies for "east/west" traffic and protection from compromised sidecar proxies by defining service level security policies. Another functionality that Cilium provides is collecting the visibility of the traffic that is done by its unix domain socket monitor and the cilium API. The cilium API is provided by the cilium agent. So with Cilium we can collect the data between the microservices, but with the following tool (Hubble[10]) we can observe it and get deep visibility.

Hubble
Hubble is a networking and security observability platform for cloud native workloads. It requires Cilium and extended Berkeley Packet Filter. As mentioned before, it enables deep visibility into the communication and behavior of services and networking infrastructure. Hubble gets the traffic from Cilium by reading the Cilium unix domain socket monitor and the Cilium API. When Cilium is integrated with Istio, the data between microservices is encrypted by using mutual TLS as mentioned before. However, Hubble can get L7 visibility by extracting from within the Istio sidecar and thus can see the unencrypted traffic.

4 Methodology

4.1 Approach

Based on the related work, implementation of Kubernetes in combination with Istio is a big part of the Zero Trust model, but this combination can only provide visibility on the ingress and egress traffic of a node and not the communication between two containers within a node. Traffic between microservices needs to be observed in case of an attack or a service failure. This makes the administrators' life much easier in finding and tracing where the problem arose.

With a Proof of Concept we determined that with the combination of the following appropriate tools one can achieve encryption of transit data and deep traffic visibility between microservices:

- Google Cloud Platform
- Google Kubernetes Engine
- Istio
- Cilium
- Hubble

This setup was built in Google Cloud Platform (GCP) using Google Kubernetes Engine (GKE), where we created one cluster with four nodes using a machine type "n1-standard-2". Cilium was deployed as a Container Network Interface (CNI) plugin to provide networking, security, and loadbalancing. Cilium also is able to collect the data between microservices as mentioned in section 3. Istio was then deployed on top of Cilium providing micro-segmentation with the help of sidecar proxies by having the appropriate authorization policies, as well as encryption of transit data between microservices using mutual TLS (mTLS). Finally we deployed Hubble on top of Istio. The combination of eBPF, Cilium,

Istio and Hubble will give us the ability to enable deep visibility into the communication and behavior of the microservices, as well as protection to the data that are being transferred between them. The whole setup is depicted in figure 2.
It is important to prove that the network traffic between two containers was visible. In order to achieve this we deployed a demo application to have traffic flowing between microservices so we could visualize it.

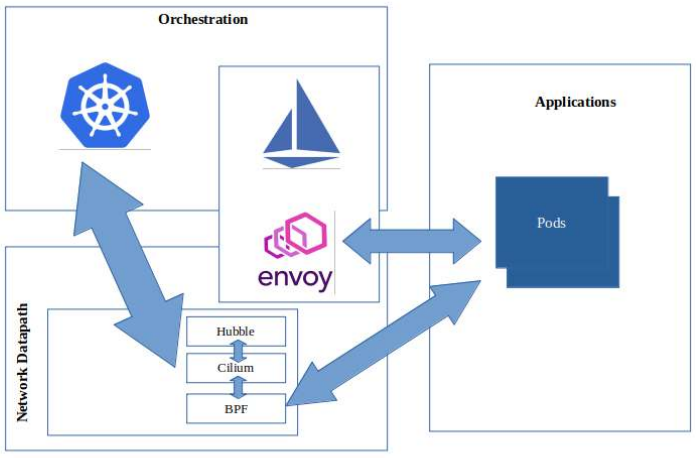


Figure 2: Implementation Setup.

4.2 Scope

The general scope of this project is to mitigate the data leakage within a containerized environment. In order to achieve this, an actionable traffic visualization needs to be accomplished as well as encryption when the data are transferred between containers. By having reasonable traffic logging, we can have a proper analysis by tracing and finding where and when the attack occurred and by who.
Thus, the scope of our research was limited to the operational level[18] and more precisely, providing deep visibility of the traffic that occurs between microservices (east-west) and implementing encryption of the transit data. This can be achieved by using the appropriate tools that were mentioned previously in section 3.

5 Results

This section describes the results of our research. In order to have viable results we deployed a demo application, which is illustrated in figure 3. The traffic we monitored for our Proof of Concept was the communication between the "product-page" and the "reviews-v1" sidecar proxies.

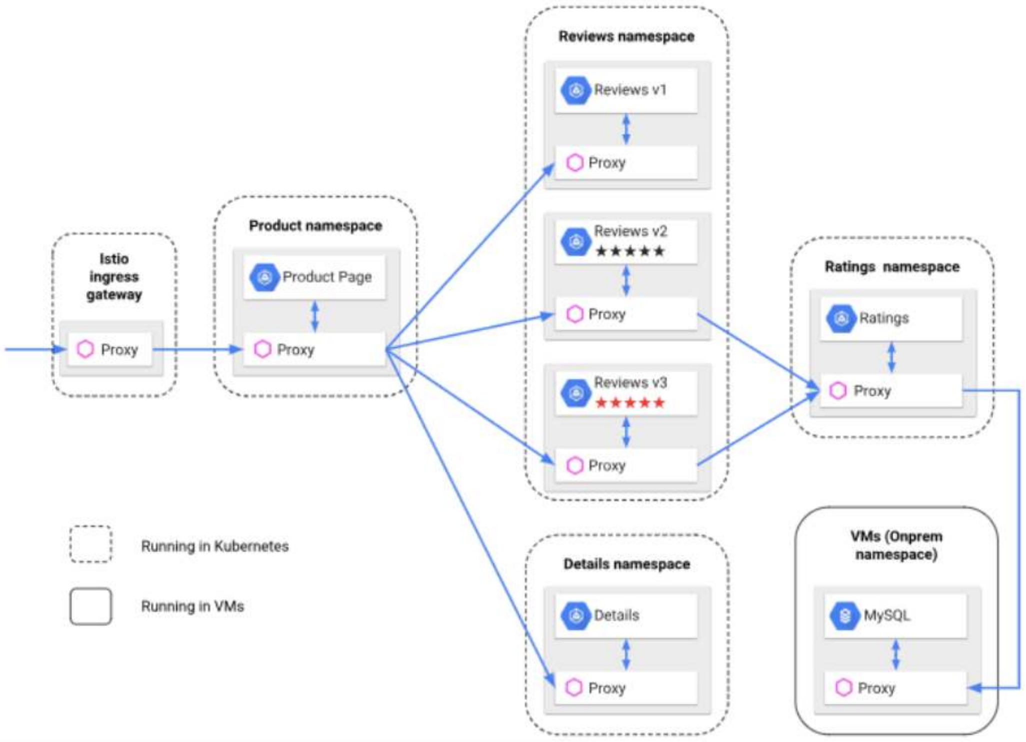
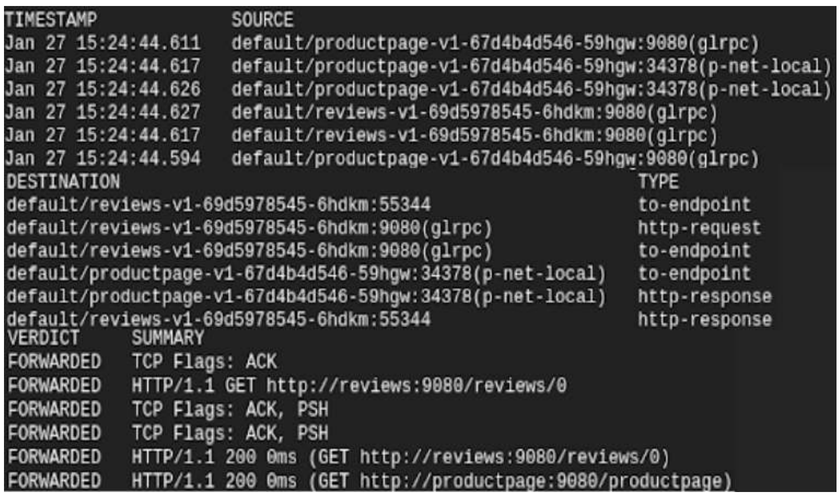


Figure 3: Demo Application.

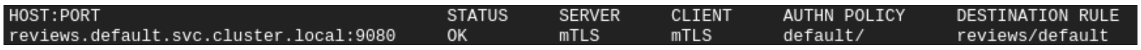
Since Hubble is observing the traffic between microservices inside the envoy proxy , we are able to visualize the following traffic data unencrypted: the timestamp, source, destination, type, verdict and a summary about the packet. This is depicted in figure 4. In figure 4 we can observe the http request from the "product-page" pod to the "reviews-v1" pod, as well as the response from the "review-v1" pod to the "product-page" pod.



```
TIMESTAMP            SOURCE
Jan 27 15:24:44.611  default/productpage-v1-67d4b4d546-59hgw:9080(glrpc)
Jan 27 15:24:44.617  default/productpage-v1-67d4b4d546-59hgw:34378(p-net-local)
Jan 27 15:24:44.626  default/productpage-v1-67d4b4d546-59hgw:34378(p-net-local)
Jan 27 15:24:44.627  default/reviews-v1-69d5978545-6hdkm:9080(glrpc)
Jan 27 15:24:44.617  default/reviews-v1-69d5978545-6hdkm:9080(glrpc)
Jan 27 15:24:44.594  default/productpage-v1-67d4b4d546-59hgw:9080(glrpc)
DESTINATION                                              TYPE
default/reviews-v1-69d5978545-6hdkm:55344                to-endpoint
default/reviews-v1-69d5978545-6hdkm:9080(glrpc)          http-request
default/reviews-v1-69d5978545-6hdkm:9080(glrpc)          to-endpoint
default/productpage-v1-67d4b4d546-59hgw:34378(p-net-local)  to-endpoint
default/productpage-v1-67d4b4d546-59hgw:34378(p-net-local)  http-response
default/reviews-v1-69d5978545-6hdkm:55344                http-response
VERDICT     SUMMARY
FORWARDED   TCP Flags: ACK
FORWARDED   HTTP/1.1 GET http://reviews:9080/reviews/0
FORWARDED   TCP Flags: ACK, PSH
FORWARDED   TCP Flags: ACK, PSH
FORWARDED   HTTP/1.1 200 0ms (GET http://reviews:9080/reviews/0)
FORWARDED   HTTP/1.1 200 0ms (GET http://productpage:9080/productpage)
```

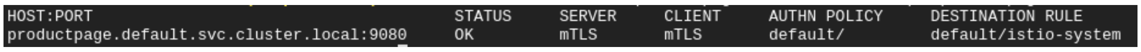
Figure 4: Traffic captured with Hubble.

We know that with the help of Istio we can have encrypted traffic flowing between the microservices. In order to check that encryption (mutual TLS) was indeed implemented we used a function that Istio provides to check if the mTLS was enabled and operating. In figures 5 and 6 we can see that mutual TLS was indeed operating on both services. However, this is not enough to verify that mutual TLS was enabled, so we needed to sniff the traffic between the microservices. We did this by executing tcpdump[9] in the proxy sidecar container of the "product-page" and sending requests from the "reviews-v1" to the "product-page". In figures 7 and 8 one can see how we've done it and in figure 9 one can see the part of the output of the tcpdump that includes the packets originated from "review-v1". As expected, one cannot see any detail about the communication since it happened through TLS.



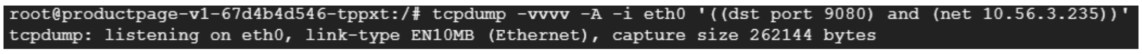
```
HOST:PORT                                   STATUS     SERVER     CLIENT     AUTHN POLICY     DESTINATION RULE
reviews.default.svc.cluster.local:9080      OK         mTLS       mTLS       default/         reviews/default
```

Figure 5: Review-v1 pod mTLS encryption.



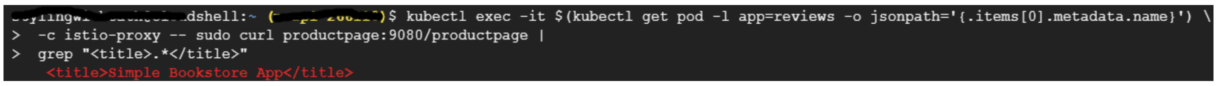
```
HOST:PORT                                      STATUS     SERVER     CLIENT     AUTHN POLICY     DESTINATION RULE
productpage.default.svc.cluster.local:9080     OK         mTLS       mTLS       default/         default/istio-system
```

Figure 6: Product-page pod mTLS encryption.



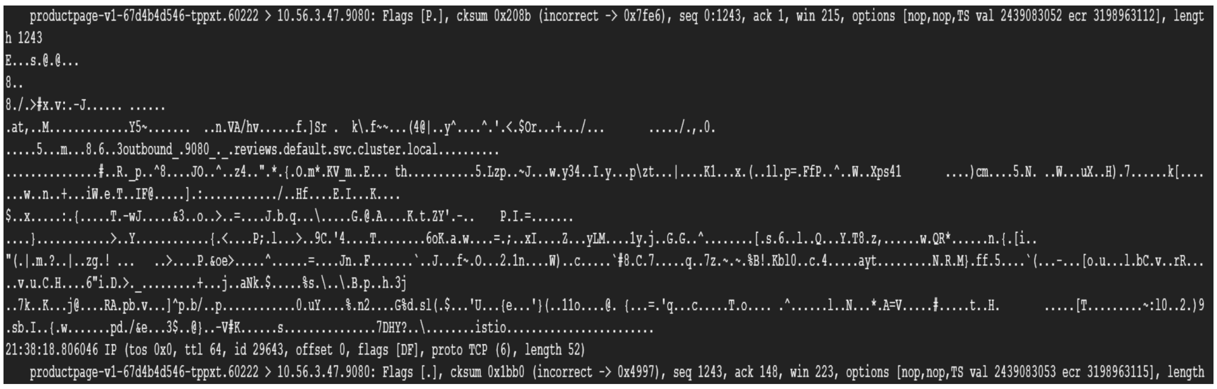
```
root@productpage-v1-67d4b4d546-tppxt:/# tcpdump -vvvv -A -i eth0 '((dst port 9080) and (net 10.56.3.235))'
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Figure 7: Tcpdump command executed in sidecar proxy of product-page.



```
[redacted]dshell:~ ([redacted])$ kubectl exec -it $(kubectl get pod -l app=reviews -o jsonpath='{.items[0].metadata.name}') \
>  -c istio-proxy -- sudo curl productpage:9080/productpage |
>  grep "<title>.*</title>"
    <title>Simple Bookstore App</title>
```

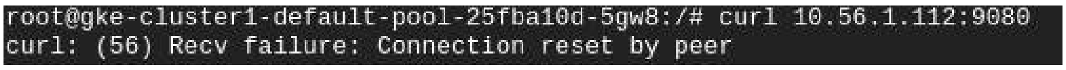
Figure 8: Sending http request from reviews-v1 pod to product-page pod.



```
    productpage-v1-67d4b4d546-tppxt.60222 > 10.56.3.47.9080: Flags [P.], cksum 0x208b (incorrect -> 0x7fe6), seq 0:1243, ack 1, win 215, options [nop,nop,TS val 2439083052 ecr 3198963112], lengt
h 1243
E...s.@.@...
8..
8./.>#x.v:.-J...... ......
.at,..M............Y5~.......  ..n.VA/hv......f.]Sr .  k\.f~~...(4@|..y^....^.'.<.$Or...+.../...       ...../.,.0.
.....5...m...8.6..3outbound_.9080_._.reviews.default.svc.cluster.local.........
..............#..R._p..^8....JO..^..z4..".*.{.O.m*.KV_m..E... th...........5.Lzp..~J...w.y34..I.y...p\zt...|....K1...x.(..1l.p=.FfP..^..W..Xps41       ....)cm....5.N. ..W...uX..H).7......k[....
...w..n..+...iW.e.T..IF@.....].:.........../..Hf....E.I...K....
$..x.....:.{.....T.-wJ.....&3..o..>..=....J.b.q...\.....G.@.A....K.t.ZY'.-..    P.I.=.......
....}...........>..Y...........{.<....P;.l...>..9C.'4....T........6oK.a.w....=.;..xI....Z...yLM....1y.j..G.G..^........[.s.6..1..Q...Y.T8.z,......w.QR*......n.{.[i..
"(.|.m.?..|..zg.! ...   ..>....P.&oe>.....^......=....Jn..F.......`..J...f~.O...2.1n....W)..c.....`#8.C.7.....q..7z.~.~.%B!.Kbl0..c.4.....ayt.........N.R.M}.ff.5....`(...-...[o.u...l.bC.v..rR...
..v.u.C.H....6"i.D.>._.........+...j..aNk.$.....%s.\..\.B.p..h.3j
..7k..K...j@....RA.pb.v...]^p.b/..p...........0.uY....%.n2....G%d.sl(.$...'U...{e...'}(..1lo....@. {...=.'q...c.....T.o.... .^......l..N...*.A=V.....#.....t..H.       .....[T.........~:10..2.)9
.sb.I..{.w.......pd./&e...3$..@}..-V#K......s...............7DHY?..\........istio.......................
21:38:18.806046 IP (tos 0x0, ttl 64, id 29643, offset 0, flags [DF], proto TCP (6), length 52)
    productpage-v1-67d4b4d546-tppxt.60222 > 10.56.3.47.9080: Flags [.], cksum 0x1bb0 (incorrect -> 0x4997), seq 1243, ack 148, win 223, options [nop,nop,TS val 2439083053 ecr 3198963115], length
```

Figure 9: Output of tcpdump is encrypted because of TLS.

Finally the last results we got from this research is micro-segmentation. We deployed a container in our cluster, which is not part of the Istio service mesh in order to try and send out requests to one of the containers that is implemented with Istio service mesh. The container we chose is "reviews-v1", which has an IP address 10.56.1.112. We tried to send a http request to "reviews-v1" container from the new container that we deployed. Since the new container that we deployed is not part of the Istio service mesh and not part of the application, the communication could not be established. This is illustrated in figure 10.



```
root@gke-cluster1-default-pool-25fba10d-5gw8:/# curl 10.56.1.112:9080
curl: (56) Recv failure: Connection reset by peer
```

Figure 10: Sending http request from a non service mesh container.

6 Discussion

In this section we discuss the following topics.

6.1 Operational controls

In this subsection we discuss what are the operational controls that are used by the tools we implemented to achieve our goal.
First, Istio service mesh tool provides Secure Sockets Layer (SSL) encryption for "east-west" traffic as well as for "north-south". Considering that there is an ingress and egress sidecar proxy for the traffic flowing from inside the service mesh to outside and vice versa we can have "north-south" SSL encryption. Another operational control that Istio provides to our project is that Istio as mentioned in the background section consists of a control plane that is a set of centrally-managed services that operates independently of the applications running within the service mesh. Furthermore, micro-segmentation can be achieved by adding the appropriate authorization policies to the sidecar proxies. Lastly, since there is ingress and egress sidecar proxies to monitor traffic flowing in and out of the service mesh, we can apply restrictive inbound and outbound access for our service mesh.
For the Cilium tool in our setup, is for enhancing the network security rules by loadbalancing the traffic or preventing traffic to flow to specific containers.
On the other hand, Hubble, which operates in combination with eBPF and Cilium, provides data classification by separating traffic via protocols, as well as it provides us deep observation in the traffic, which is flowing between the microservices. Hubble also gives us the ability for deep visibility into the behavior of services as well as the networking infrastructure in a completely transparent manner.

6.2 Compromised sidecar proxies

As mentioned before Hubble reads the unencrypted traffic from within the sidecar proxy. To access the sidecar proxy container one need root privileges. Now let's assume that the sidecar proxy is compromised. To mitigate capturing of the unencrypted traffic, this sidecar proxy needs to operate with least privileges. This can be achieved by using Cilium in addition to Istio by defining service level security policies.

6.3 Hubble used in practice

Hubble is a fairly new platform; the first version was released in December 2019. It is still in beta stage: new functionalities are being added like automation, etc. Thus Hubble can be implemented in a Proof of Concept, but since it's not in a stable, production stage, it cannot be implemented in practice yet.

7 Conclusion

In this research, the following research question was answered: *How to implement Zero Trust for "east/west" traffic between microservices in containerized environment.* In order to answer our research question, we first needed to answer the following sub-questions: *How to regulate the "east-west" traffic flow?* and *How to implement confidentiality for transit data?*. To answer our sub-questions we have performed research regarding the operational controls (figure 11) that were required to be implemented in order to achieve Zero Trust in containerized environment.
The implementation of Kubernetes and Istio service mesh provides a big part of the Zero Trust model in the containerized environment, but there is an important piece that was missing from the implementation, which is deep traffic visibility between microservices. In order to cope with this problem, another tool was needed to be implemented, which is Cilium, and which comes with extended Berkeley Packet Filter (eBPF). Cilium is a networking plugin that integrates with Istio and enhances the security and performance of Istio, as well as collecting traffic between microservices. Lastly, since we're limited to the fact that Cilium is only collecting the data between microservices, but we wanted to observe the traffic as well, Hubble needed to be implemented, which enables deep visibility into the communication and behavior of services and networking infrastructure.
By installing the above tools we were able to regulate the traffic by implementing micro-segmentation while adding the appropriate authorization policies to the sidecar proxies, as well as having encrypted communication between them by using mutual TLS. We also managed to get deep visibility of the communication between microservices ("east-west" traffic). To support our solution we've implemented a Proof of Concept. In conclusion, with these tools we are able to apply some of the required Zero Trust operational controls to mitigate data leakage in a containerized environment.

8 Future Work

8.1 Data leakage detection

To mitigate data leakage, one also need to detect it. So an interesting future work could be is to implement data leakage detection by applying data leakage detection protection controls.

8.2 Behavioral Analytics

An interesting operational control to mitigate data leakage is behavioral analytics. With this control one can detect abnormalities on normal traffic flows. So to implement behavioral analytics could be another future work.

8.3 Content-Inspection

Inspecting the content of the traffic to detect intruders and prevent them to cause data leakage can also be an interesting future work. We can achieve this by adding Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS).

References

[1] Envoy Project Authors. *Envoy Proxy*. URL: `https://www.envoyproxy.io/`.

[2] Istio authors. *secure naming*. URL: `https://istio.io/docs/concepts/security/#secure-naming`.

[3] Cilium. *Cilium*. URL: `https://cilium.io/`.

[4] Cilium. *How Cilium enhances Istio with socket-aware BPF programs*. URL: `https://cilium.io/blog/2018/08/07/istio-10-cilium/`.

[5] Cilium. *Istio*. URL: `https://cilium.io/blog/2018/08/07/istio-10-cilium/`.

[6] Casimer DeCusatis et al. "Implementing zero trust cloud networks with transport access control and first packet authentication". In: *2016 IEEE International Conference on Smart Cloud (SmartCloud)*. IEEE. 2016, pp. 5–10.

[7] Cilium Docs. *BPF and XDP Reference Guide*. URL: `https://docs.cilium.io/en/v1.6/bpf/`.

[8] GoDocs. *package sockmap*. URL: `https://godoc.org/github.com/cilium/cilium/pkg/maps/sockmap`.

[9] The Tcpdump Group. *tcpdum manpage*. URL: `https://www.tcpdump.org/manpages/tcpdump.1.html`.

[10] Hubble. *Hubble*. URL: `https://github.com/cilium/hubble`.

[11] Fatima Hussain et al. "Intelligent Service Mesh Framework for API Security and Management". In: *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2019, pp. 0735–0742.

[12] *Istio*. 2018. URL: `https://istio.io/`.

[13] Michael Kerrisk. *Linux Programmer's Manual BPF(2)*. URL: `http://man7.org/linux/man-pages/man2/bpf.2.html`.

[14] *Kubernetes*. URL: `https://kubernetes.io/`.

[15] Kubernetes. *Kubernetes*. URL: `https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/`.

[16] ON2IT. *Cloud container security*. [Online; accessed 6-January-2020]. URL: `https://on2it.net/en/cloud-security/container-security/`.

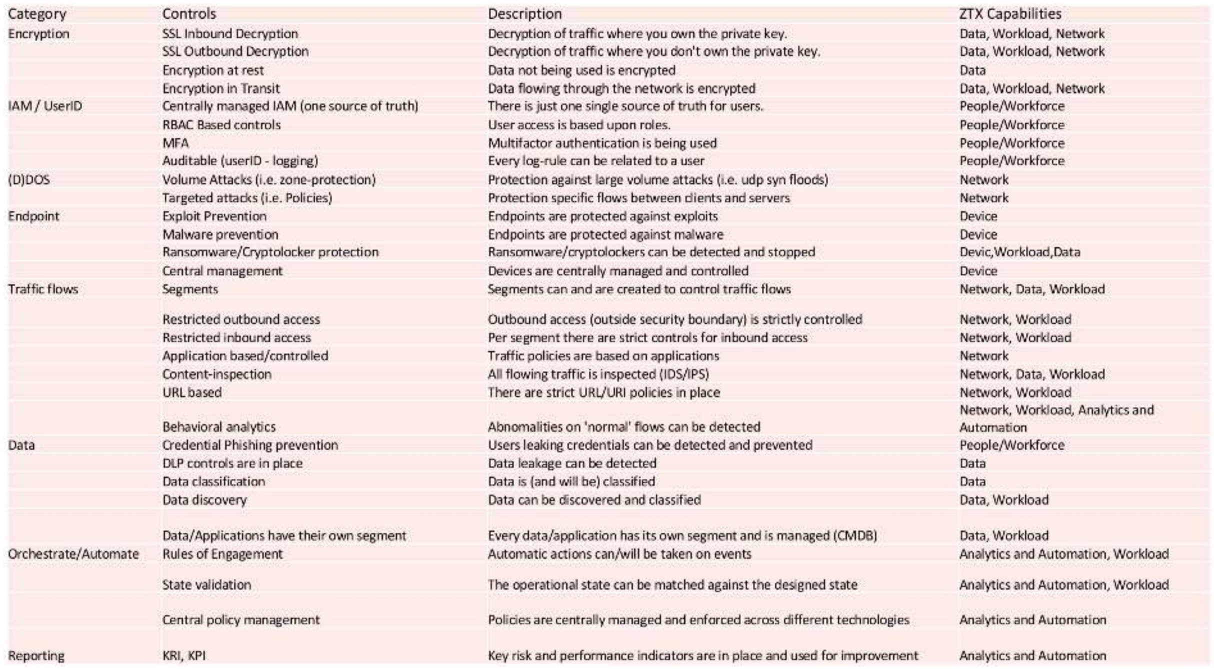
[17] ON2IT. *Zero Trust*. [Online; accessed 28-January-2020]. URL: `https://on2it.net/en/zero-trust/`.

[18] ON2IT. *Zero Trust Security Framework*. [Online; accessed 6-January-2020]. URL: `https://on2it.net/en/zero-trust-security-framework/`.

[19] Limin Wang. *Micro-Segmentation with Istio Authorization*. URL: `https://istio.io/blog/2018/istio-authorization/`.

[20] Zirak Zaheer et al. "eZTrust: Network-Independent Zero-Trust Perimeterization for Microservices". In: *Proceedings of the 2019 ACM Symposium on SDN Research*. 2019, pp. 49–61.

9 Appendix



Category	Controls	Description	ZTX Capabilities
Encryption	SSL Inbound Decryption	Decryption of traffic where you own the private key.	Data, Workload, Network
	SSL Outbound Decryption	Decryption of traffic where you don't own the private key.	Data, Workload, Network
	Encryption at rest	Data not being used is encrypted	Data
	Encryption in Transit	Data flowing through the network is encrypted	Data, Workload, Network
IAM / UserID	Centrally managed IAM (one source of truth)	There is just one single source of truth for users.	People/Workforce
	RBAC Based controls	User access is based upon roles.	People/Workforce
	MFA	Multifactor authentication is being used	People/Workforce
	Auditable (userID - logging)	Every log-rule can be related to a user	People/Workforce
(D)DOS	Volume Attacks (i.e. zone-protection)	Protection against large volume attacks (i.e. udp syn floods)	Network
	Targeted attacks (i.e. Policies)	Protection specific flows between clients and servers	Network
Endpoint	Exploit Prevention	Endpoints are protected against exploits	Device
	Malware prevention	Endpoints are protected against malware	Device
	Ransomware/Cryptolocker protection	Ransomware/cryptolockers can be detected and stopped	Devic,Workload,Data
	Central management	Devices are centrally managed and controlled	Device
Traffic flows	Segments	Segments can and are created to control traffic flows	Network, Data, Workload
	Restricted outbound access	Outbound access (outside security boundary) is strictly controlled	Network, Workload
	Restricted inbound access	Per segment there are strict controls for inbound access	Network, Workload
	Application based/controlled	Traffic policies are based on applications	Network
	Content-inspection	All flowing traffic is inspected (IDS/IPS)	Network, Data, Workload
	URL based	There are strict URL/URI policies in place	Network, Workload
	Behavioral analytics	Abnomalities on 'normal' flows can be detected	Network, Workload, Analytics and Automation
Data	Credential Phishing prevention	Users leaking credentials can be detected and prevented	People/Workforce
	DLP controls are in place	Data leakage can be detected	Data
	Data classification	Data is (and will be) classified	Data
	Data discovery	Data can be discovered and classified	Data, Workload
	Data/Applications have their own segment	Every data/application has its own segment and is managed (CMDB)	Data, Workload
Orchestrate/Automate	Rules of Engagement	Automatic actions can/will be taken on events	Analytics and Automation, Workload
	State validation	The operational state can be matched against the designed state	Analytics and Automation, Workload
	Central policy management	Policies are centrally managed and enforced across different technologies	Analytics and Automation
Reporting	KRI, KPI	Key risk and performance indicators are in place and used for improvement	Analytics and Automation

Figure 11: Operational controls.