TECHNISCHE UNIVERSITÄT DRESDEN

Dominik Pataky

Faculty of Computer Science, Institute of Systems Architecture, Chair of Computer Networks

Tencrypt: Encrypting Tenant-Traffic in OpenShift

Forschungsprojekt Anwendung // Dresden, 15th November, 2018

Contents

Introduction

Red Hat OpenShift
- Networking, Security

Security requirements and threat model

Encrypting traffic between Pods
- Fundamentals, ideas and possible approaches
- Using Minishift for experimental implementations
- Implementation concepts
- Proof of concept implementation
- Throughput measurements

Conclusion

Overview

- Virtualisation of computing resources is a trending and advancing topic
- In the recent years, virtualisation and emulation of hardware („virtual machines“) was steadily replaced by containerisation
- Containerisation uses different techniques to isolate applications running on the same kernel, saving the emulation overhead
- Whole ecosystems revolve around these engines (e.g. Docker), enabling faster deployment and development

Overview

- Virtualisation of computing resources is a trending and advancing topic
- In the recent years, virtualisation and emulation of hardware („virtual machines“) was steadily replaced by containerisation
- Containerisation uses different techniques to isolate applications running on the same kernel, saving the emulation overhead
- Whole ecosystems revolve around these engines (e.g. Docker), enabling faster deployment and development

Overview

- Virtualisation of computing resources is a trending and advancing topic
- In the recent years, virtualisation and emulation of hardware („virtual machines“) was steadily replaced by containerisation
- Containerisation uses different techniques to isolate applications running on the same kernel, saving the emulation overhead
- Whole ecosystems revolve around these engines (e.g. Docker), enabling faster deployment and development

Overview

- Virtualisation of computing resources is a trending and advancing topic
- In the recent years, virtualisation and emulation of hardware („virtual machines“) was steadily replaced by containerisation
- Containerisation uses different techniques to isolate applications running on the same kernel, saving the emulation overhead
- Whole ecosystems revolve around these engines (e.g. Docker), enabling faster deployment and development

Tencrypt

1. Explores possibilities for transparent encryption of traffic between Pods of the same Project (Tenants)
2. Deep-dive into OpenShift and related technologies
3. Security requirements and anticipated threat model
4. Collection of ideas and possible approaches
5. Evaluating Minishift as development environment
6. Implementation concept parts
7. Proof of concept implementation in Go
8. Throughput measurements and conclusion

Tencrypt

1. Explores possibilities for transparent encryption of traffic between Pods of the same Project (Tenants)
2. Deep-dive into OpenShift and related technologies
3. Security requirements and anticipated threat model
4. Collection of ideas and possible approaches
5. Evaluating Minishift as development environment
6. Implementation concept parts
7. Proof of concept implementation in Go
8. Throughput measurements and conclusion

Tencrypt

1. Explores possibilities for transparent encryption of traffic between Pods of the same Project (Tenants)
2. Deep-dive into OpenShift and related technologies
3. Security requirements and anticipated threat model
4. Collection of ideas and possible approaches
5. Evaluating Minishift as development environment
6. Implementation concept parts
7. Proof of concept implementation in Go
8. Throughput measurements and conclusion

Tencrypt

1. Explores possibilities for transparent encryption of traffic between Pods of the same Project (Tenants)
2. Deep-dive into OpenShift and related technologies
3. Security requirements and anticipated threat model
4. Collection of ideas and possible approaches
5. Evaluating Minishift as development environment
6. Implementation concept parts
7. Proof of concept implementation in Go
8. Throughput measurements and conclusion

Tencrypt

1. Explores possibilities for transparent encryption of traffic between Pods of the same Project (Tenants)
2. Deep-dive into OpenShift and related technologies
3. Security requirements and anticipated threat model
4. Collection of ideas and possible approaches
5. Evaluating Minishift as development environment
6. Implementation concept parts
7. Proof of concept implementation in Go
8. Throughput measurements and conclusion

Tencrypt

1. Explores possibilities for transparent encryption of traffic between Pods of the same Project (Tenants)
2. Deep-dive into OpenShift and related technologies
3. Security requirements and anticipated threat model
4. Collection of ideas and possible approaches
5. Evaluating Minishift as development environment
6. Implementation concept parts
7. Proof of concept implementation in Go
8. Throughput measurements and conclusion

Tencrypt

1. Explores possibilities for transparent encryption of traffic between Pods of the same Project (Tenants)
2. Deep-dive into OpenShift and related technologies
3. Security requirements and anticipated threat model
4. Collection of ideas and possible approaches
5. Evaluating Minishift as development environment
6. Implementation concept parts
7. Proof of concept implementation in Go
8. Throughput measurements and conclusion

Tencrypt

1. Explores possibilities for transparent encryption of traffic between Pods of the same Project (Tenants)
2. Deep-dive into OpenShift and related technologies
3. Security requirements and anticipated threat model
4. Collection of ideas and possible approaches
5. Evaluating Minishift as development environment
6. Implementation concept parts
7. Proof of concept implementation in Go
8. Throughput measurements and conclusion

OpenShift, Kubernetes and Docker

- **Docker**: wrapper for Linux kernel namespaces and cgroup features. Introduces features like reproducible images (Dockerfiles), image registries and toolchain
- **Kubernetes**: uses Docker as containerisation engine for multi-node application deployment
- **OpenShift**: uses Kubernetes as the app orchestration engine, adding more features for a smoother workflow

- Not mentioned: multiple other APIs, engines and projects with similar toolchains and goals

OpenShift, Kubernetes and Docker

- **Docker**: wrapper for Linux kernel namespaces and cgroup features. Introduces features like reproducible images (Dockerfiles), image registries and toolchain
- **Kubernetes**: uses Docker as containerisation engine for multi-node application deployment
- **OpenShift**: uses Kubernetes as the app orchestration engine, adding more features for a smoother workflow

- Not mentioned: multiple other APIs, engines and projects with similar toolchains and goals

OpenShift, Kubernetes and Docker

- **Docker**: wrapper for Linux kernel namespaces and cgroup features. Introduces features like reproducible images (Dockerfiles), image registries and toolchain
- **Kubernetes**: uses Docker as containerisation engine for multi-node application deployment
- **OpenShift**: uses Kubernetes as the app orchestration engine, adding more features for a smoother workflow

- Not mentioned: multiple other APIs, engines and projects with similar toolchains and goals

OpenShift, Kubernetes and Docker

- **Docker**: wrapper for Linux kernel namespaces and cgroup features. Introduces features like reproducible images (Dockerfiles), image registries and toolchain
- **Kubernetes**: uses Docker as containerisation engine for multi-node application deployment
- **OpenShift**: uses Kubernetes as the app orchestration engine, adding more features for a smoother workflow

---

- Not mentioned: multiple other APIs, engines and projects with similar toolchains and goals

What is Red Hat OpenShift?

- Available as open source project OKD (formerly Origin)
- Supported instances by Red Hat as „Online“, „Dedicated“ or „Container Platform“
- Hardware resources called **Nodes** connect to **Master** and host **Pods**

What is Red Hat OpenShift?

- Available as open source project OKD (formerly Origin)
- Supported instances by Red Hat as „Online“, „Dedicated“ or „Container Platform“
- Hardware resources called **Nodes** connect to **Master** and host **Pods**

What is Red Hat OpenShift?

- Available as open source project OKD (formerly Origin)
- Supported instances by Red Hat as „Online“, „Dedicated“ or „Container Platform“
- Hardware resources called **Nodes** connect to **Master** and host **Pods**

Related tools Related technologies

Components of OpenShift

- **Nodes** (RHEL)
- **Master** (APIs, Authentication, Storage, Scheduling, Scaling)
- **Pods** (grouping of Containers, Users/Projects, Policies)
- Container image **Registry**
- **Persistent Storage** (Volumes, NFS/GlusterFS/Ceph/Clouds)
- **Service Layer** with Service Discovery (Load-Balancing, virtual IPs)
- **Routing Layer** (HAProxy, routing external access, egress routing, A/B testing)

Components of OpenShift

- **Nodes** (RHEL)
- **Master** (APIs, Authentication, Storage, Scheduling, Scaling)
- **Pods** (grouping of Containers, Users/Projects, Policies)
- Container image **Registry**
- **Persistent Storage** (Volumes, NFS/GlusterFS/Ceph/Clouds)
- **Service Layer** with Service Discovery (Load-Balancing, virtual IPs)
- **Routing Layer** (HAProxy, routing external access, egress routing, A/B testing)

Components of OpenShift

- **Nodes** (RHEL)
- **Master** (APIs, Authentication, Storage, Scheduling, Scaling)
- **Pods** (grouping of Containers, Users/Projects, Policies)
- Container image **Registry**
- **Persistent Storage** (Volumes, NFS/GlusterFS/Ceph/Clouds)
- **Service Layer** with Service Discovery (Load-Balancing, virtual IPs)
- **Routing Layer** (HAProxy, routing external access, egress routing, A/B testing)

Components of OpenShift

- **Nodes** (RHEL)
- **Master** (APIs, Authentication, Storage, Scheduling, Scaling)
- **Pods** (grouping of Containers, Users/Projects, Policies)
- Container image **Registry**
- **Persistent Storage** (Volumes, NFS/GlusterFS/Ceph/Clouds)
- **Service Layer** with Service Discovery (Load-Balancing, virtual IPs)
- **Routing Layer** (HAProxy, routing external access, egress routing, A/B testing)

Components of OpenShift

- **Nodes** (RHEL)
- **Master** (APIs, Authentication, Storage, Scheduling, Scaling)
- **Pods** (grouping of Containers, Users/Projects, Policies)
- Container image **Registry**
- **Persistent Storage** (Volumes, NFS/GlusterFS/Ceph/Clouds)
- **Service Layer** with Service Discovery (Load-Balancing, virtual IPs)
- **Routing Layer** (HAProxy, routing external access, egress routing, A/B testing)

Components of OpenShift

- **Nodes** (RHEL)
- **Master** (APIs, Authentication, Storage, Scheduling, Scaling)
- **Pods** (grouping of Containers, Users/Projects, Policies)
- Container image **Registry**
- **Persistent Storage** (Volumes, NFS/GlusterFS/Ceph/Clouds)
- **Service Layer** with Service Discovery (Load-Balancing, virtual IPs)
- **Routing Layer** (HAProxy, routing external access, egress routing, A/B testing)

Components of OpenShift

- **Nodes** (RHEL)
- **Master** (APIs, Authentication, Storage, Scheduling, Scaling)
- **Pods** (grouping of Containers, Users/Projects, Policies)
- Container image **Registry**
- **Persistent Storage** (Volumes, NFS/GlusterFS/Ceph/Clouds)
- **Service Layer** with Service Discovery (Load-Balancing, virtual IPs)
- **Routing Layer** (HAProxy, routing external access, egress routing, A/B testing)

Networking in OpenShift

- Internal DNS servers for Services
- Split DNS with SkyDNS
- Container Networking Interface (CNI)
- Software Defined Networking (SDN)
  - Flat network: all Pods reach each other
  - Multi-Tenant: isolated traffic on Project level by Virtual Network ID (VNID)
  - Network policy: granular policy rules for Projects & Pods
- VXLAN overlay for Pod-to-Pod by Open vSwitch (OVS)

Networking in OpenShift

- Internal DNS servers for Services
- Split DNS with SkyDNS
- Container Networking Interface (CNI)
- Software Defined Networking (SDN)
  - Flat network: all Pods reach each other
  - Multi-Tenant: isolated traffic on Project-level by Virtual Network ID (VNID)
  - Network policy: granular policy-rules for Projects/Pods
- VXLAN overlay for Pod-to-Pod by Open vSwitch (OVS)

Networking in OpenShift

- Internal DNS servers for Services
- Split DNS with SkyDNS
- Container Networking Interface (CNI)
- Software Defined Networking (SDN)
  - Flat network: all Pods reach each other
  - Multi-Tenant: isolated traffic on Project-level by Virtual Network ID (VNID)
  - Network policy: granular policy-rules for Projects/Pods
- VXLAN overlay for Pod-to-Pod by Open vSwitch (OVS)

Networking in OpenShift

- Internal DNS servers for Services
- Split DNS with SkyDNS
- Container Networking Interface (CNI)
- Software Defined Networking (SDN)
  - Flat network: all Pods reach each other
  - Multi-Tenant: isolated traffic on Project-level by Virtual Network ID (VNID)
  - Network policy: granular policy-rules for Projects/Pods
- VXLAN overlay for Pod-to-Pod by Open vSwitch (OVS)

Networking in OpenShift

- Internal DNS servers for Services
- Split DNS with SkyDNS
- Container Networking Interface (CNI)
- Software Defined Networking (SDN)
  - Flat network: all Pods reach each other
  - Multi-Tenant: isolated traffic on Project-level by Virtual Network ID (VNID)
  - Network policy: granular policy-rules for Projects/Pods
- VXLAN overlay for Pod-to-Pod by Open vSwitch (OVS)

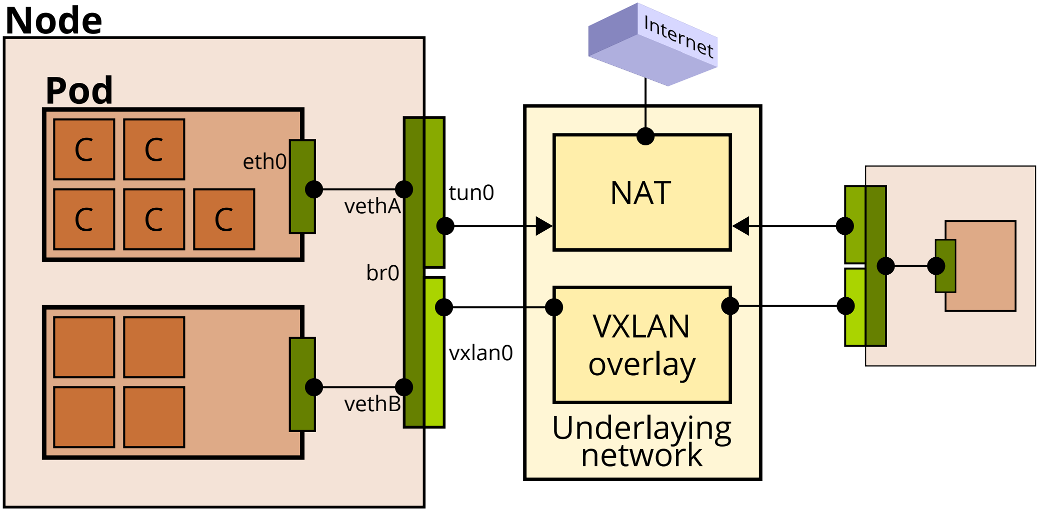

Figure 1: Overview of networking of Nodes, Pods and Containers

Security in OpenShift

- Policies for Container deployment
- Multi-tenancy through Users and Projects
- Container host pinning
- Additional mechanisms to secure Container image creation
- Secret Management through secured storage in Master
  - Access within Pods through ENV or mounts

Security in OpenShift

- Policies for Container deployment
- Multi-tenancy through Users and Projects
- Container host pinning
- Additional mechanisms to secure Container image creation
- Secret Management through secured storage in Master
  - Access within Pods through ENV or mounts

Security in OpenShift

- Policies for Container deployment
- Multi-tenancy through Users and Projects
- Container host pinning
- Additional mechanisms to secure Container image creation
- Secret Management through secured storage in Master
  - Access within Pods through ENV or mounts

Security in OpenShift

- Policies for Container deployment
- Multi-tenancy through Users and Projects
- Container host pinning
- Additional mechanisms to secure Container image creation
- Secret Management through secured storage in Master
  - Access within Pods through ENV or mounts

Security in OpenShift

- Policies for Container deployment
- Multi-tenancy through Users and Projects
- Container host pinning
- Additional mechanisms to secure Container image creation
- Secret Management through secured storage in Master
  - Access within Pods through ENV or mounts

Security requirements

- **Authentication**: Pods must be able to ensure sender authenticity
- **Integrity**: Transmitted data must not be corrupted or manipulated
- **Confidentiality**: Pod-to-Pod traffic must not be readable by third parties
- **Availability**: Key concept in underlying Kubernetes engine, which Tencrypt must not interfere with
- **Authorisation**: Pods should reject unencrypted traffic from internal Pods

- Based on STRIDE/AINCAA given in [Sho14].
- Introduced in PoC: Confidentiality

Security requirements

- **Authentication**: Pods must be able to ensure sender authenticity
- **Integrity**: Transmitted data must not be corrupted or manipulated
- **Confidentiality**: Pod-to-Pod traffic must not be readable by third parties
- **Availability**: Key concept in underlying Kubernetes engine, which Tencrypt must not interfere with
- **Authorisation**: Pods should reject unencrypted traffic from internal Pods

- Based on STRIDE/AINCAA given in [Sho14].
- Introduced in PoC: Confidentiality

Security requirements

- **Authentication**: Pods must be able to ensure sender authenticity
- **Integrity**: Transmitted data must not be corrupted or manipulated
- **Confidentiality**: Pod-to-Pod traffic must not be readable by third parties
- **Availability**: Key concept in underlying Kubernetes engine, which Tencrypt must not interfere with
- **Authorisation**: Pods should reject unencrypted traffic from internal Pods

- Based on STRIDE/AINCAA given in [Sho14].
- Introduced in PoC: Confidentiality

Security requirements

- **Authentication**: Pods must be able to ensure sender authenticity
- **Integrity**: Transmitted data must not be corrupted or manipulated
- **Confidentiality**: Pod-to-Pod traffic must not be readable by third parties
- **Availability**: Key concept in underlying Kubernetes engine, which Tencrypt must not interfere with
- **Authorisation**: Pods should reject unencrypted traffic from internal Pods

- Based on STRIDE/AINCAA given in [Sho14].
- Introduced in PoC: Confidentiality

Security requirements

- **Authentication**: Pods must be able to ensure sender authenticity
- **Integrity**: Transmitted data must not be corrupted or manipulated
- **Confidentiality**: Pod-to-Pod traffic must not be readable by third parties
- **Availability**: Key concept in underlying Kubernetes engine, which Tencrypt must not interfere with
- **Authorisation**: Pods should reject unencrypted traffic from internal Pods

- Based on STRIDE/AINCAA given in [Sho14].
- Introduced in PoC: Confidentiality

Security requirements

- **Authentication**: Pods must be able to ensure sender authenticity
- **Integrity**: Transmitted data must not be corrupted or manipulated
- **Confidentiality**: Pod-to-Pod traffic must not be readable by third parties
- **Availability**: Key concept in underlying Kubernetes engine, which Tencrypt must not interfere with
- **Authorisation**: Pods should reject unencrypted traffic from internal Pods

---

- Based on STRIDE/AINCAA given in [Sho14].
- Introduced in PoC: Confidentiality

Security requirements

- **Authentication**: Pods must be able to ensure sender authenticity
- **Integrity**: Transmitted data must not be corrupted or manipulated
- **Confidentiality**: Pod-to-Pod traffic must not be readable by third parties
- **Availability**: Key concept in underlying Kubernetes engine, which Tencrypt must not interfere with
- **Authorisation**: Pods should reject unencrypted traffic from internal Pods

---

- Based on STRIDE/AINCAA given in [Sh014].
- Introduced in PoC: Confidentiality

Threats

ID	Description	Mitigation
T1	An attacker uses a Pod to intercept traffic originating from other namespaces (Pods) on the `br0` bridge.	Encryption of traffic, hardening of isolation mechanisms (Linux kernel).
T2	An attacker not only intercepts, but is able to modify traffic on the `br0` bridge or the `vxlan0` adapter.	Encryption and integrity checks.
T3	Interception and modification of Master-to-Node traffic.	IPsec.
T4	Interception of Node-to-Node traffic, both Project-internal and cross-Project.	A combination of Node-to-Node IPsec and Tencrypt for Project-internal traffic
T5	Incoming external Service traffic is intercepted (and maybe modified) before it reaches the handling Service namespace.	OKD Secured routes.
T6	The Pod image used by OpenShift to deploy new Pods, is maliciously modified.	Securing the image registry. *The registry depends on the used container technology and might be an external component.*
T7	Resources requested by a Pod limit the availability of other Pods on the same Node.	Continuous resource monitoring, migration or halting of resource intensive Pods if needed.

Note about containerisation

Fundamentals

- „Encrypted traffic" only includes Tenant-internal (Project-internal) traffic, not egress or cross-Project traffic
- Primary focus: `eth0` interface shared between Containers in a Pod
- At first, only application data (OSI layers 5-7) was to be encrypted. PoC encapsulates whole encrypted packet.
- Deployed container images of developers should not have to be customised at all

Fundamentals

- „Encrypted traffic“ only includes Tenant-internal (Project-internal) traffic, not egress or cross-Project traffic
- Primary focus: `eth0` interface shared between Containers in a Pod
- At first, only application data (OSI layers 5-7) was to be encrypted. PoC encapsulates whole encrypted packet.
- Deployed container images of developers should not have to be customised at all

Fundamentals

- „Encrypted traffic“ only includes Tenant-internal (Project-internal) traffic, not egress or cross-Project traffic
- Primary focus: `eth0` interface shared between Containers in a Pod
- At first, only application data (OSI layers 5-7) was to be encrypted. PoC encapsulates whole encrypted packet.
- Deployed container images of developers should not have to be customised at all

Fundamentals

- „Encrypted traffic“ only includes Tenant-internal (Project-internal) traffic, not egress or cross-Project traffic
- Primary focus: `eth0` interface shared between Containers in a Pod
- At first, only application data (OSI layers 5-7) was to be encrypted. PoC encapsulates whole encrypted packet.
- Deployed container images of developers should not have to be customised at all

Ideas and possible approaches

- Using **symmetric AES** with a shared key.
  Payload size and MTU? Rotation of keys? Fulfills security requirements?
- **Asymmetric encryption** with public keys in shared storage.
  Who generates key pair? Which system to use (e.g. X.509)? Realisable without a new OpenShift component?
- Using existing technology like **Wireguard**.
  Does it scale? Can compiled tools be integrated at all? Which component creates the interfaces? Can peer keys be shared through Secret Storage?

Ideas and possible approaches

- Using **symmetric AES** with a shared key.
  Payload size and MTU? Rotation of keys? Fulfills security requirements?
- **Asymmetric encryption** with public keys in shared storage.
  Who generates key pair? Which system to use (e.g. X.509)? Realisable without a new OpenShift component?
- Using existing technology like **Wireguard**.
  Does it scale? Can compiled tools be integrated at all? Which component creates the interfaces? Can peer keys be shared through Secret Storage?

Ideas and possible approaches

- Using **symmetric AES** with a shared key.
  Payload size and MTU? Rotation of keys? Fulfills security requirements?
- **Asymmetric encryption** with public keys in shared storage.
  Who generates key pair? Which system to use (e.g. X.509)? Realisable without a new OpenShift component?
- Using existing technology like **Wireguard**.
  Does it scale? Can compiled tools be integrated at all? Which component creates the interfaces? Can peer keys be shared through Secret Storage?

Using Minishift

- Fork of the Kubernetes Minikube project
- Development environment bundled with OKD, advertised as „local OpenShift“
- Configures a virtual machine (VirtualBox, KVM,. . .) as host for components deployed as Docker containers
- Either uses a Boot2Docker or CentOS VM ISO image
- Simulates networking and Virtual IPs (VIPs) with IPtables NAT rules
- Tencrypt: CentOS on VirtualBox, using default „developer“ account with two Projects

Using Minishift

- Fork of the Kubernetes Minikube project
- Development environment bundled with OKD, advertised as „local OpenShift“
- Configures a virtual machine (VirtualBox, KVM,. . .) as host for components deployed as Docker containers
- Either uses a Boot2Docker or CentOS VM ISO image
- Simulates networking and Virtual IPs (VIPs) with IPtables NAT rules
- Tencrypt: CentOS on VirtualBox, using default „developer“ account with two Projects

Using Minishift

- Fork of the Kubernetes Minikube project
- Development environment bundled with OKD, advertised as „local OpenShift“
- Configures a virtual machine (VirtualBox, KVM,. . .) as host for components deployed as Docker containers
- Either uses a Boot2Docker or CentOS VM ISO image
- Simulates networking and Virtual IPs (VIPs) with IPtables NAT rules
- Tencrypt: CentOS on VirtualBox, using default „developer“ account with two Projects

Using Minishift

- Fork of the Kubernetes Minikube project
- Development environment bundled with OKD, advertised as „local OpenShift“
- Configures a virtual machine (VirtualBox, KVM,. . .) as host for components deployed as Docker containers
- Either uses a Boot2Docker or CentOS VM ISO image
- Simulates networking and Virtual IPs (VIPs) with IPtables NAT rules
- Tencrypt: CentOS on VirtualBox, using default „developer“ account with two Projects

Using Minishift

- Fork of the Kubernetes Minikube project
- Development environment bundled with OKD, advertised as „local OpenShift“
- Configures a virtual machine (VirtualBox, KVM,. . .) as host for components deployed as Docker containers
- Either uses a Boot2Docker or CentOS VM ISO image
- Simulates networking and Virtual IPs (VIPs) with IPtables NAT rules
- Tencrypt: CentOS on VirtualBox, using default „developer“ account with two Projects

Using Minishift

- Fork of the Kubernetes Minikube project
- Development environment bundled with OKD, advertised as „local OpenShift“
- Configures a virtual machine (VirtualBox, KVM,. . .) as host for components deployed as Docker containers
- Either uses a Boot2Docker or CentOS VM ISO image
- Simulates networking and Virtual IPs (VIPs) with IPtables NAT rules
- Tencrypt: CentOS on VirtualBox, using default „developer“ account with two Projects

Docker image patching and Secrets

- Approach: patching the Pod image to deploy encryption proxy and custom networking
- Access to Docker daemon and images possible
  1. Re-tag original Pod image
  2. Use original image as base for patched version
  3. Build patched image with Tencrypt scripts and proxy app, tagged as „original"
- As mentioned, Secrets are either in ENV or mounts.
  Blocker: „Pod" container has no access.

Docker image patching and Secrets

- Approach: patching the Pod image to deploy encryption proxy and custom networking
- Access to Docker daemon and images possible
  1. Re-tag original Pod image
  2. Use original image as base for patched version
  3. Build patched image with Tencrypt scripts and proxy app, tagged as „original“
- As mentioned, Secrets are either in ENV or mounts.
  Blocker: „Pod“ container has no access.

Docker image patching and Secrets

- Approach: patching the Pod image to deploy encryption proxy and custom networking
- Access to Docker daemon and images possible
  1. Re-tag original Pod image
  2. Use original image as base for patched version
  3. Build patched image with Tencrypt scripts and proxy app, tagged as „original“
- As mentioned, Secrets are either in ENV or mounts.
  Blocker: „Pod“ container has no access.

Docker image patching and Secrets

- Approach: patching the Pod image to deploy encryption proxy and custom networking
- Access to Docker daemon and images possible
  1. Re-tag original Pod image
  2. Use original image as base for patched version
  3. Build patched image with Tencrypt scripts and proxy app, tagged as „original"
- As mentioned, Secrets are either in ENV or mounts.
  Blocker: „Pod" container has no access.

Docker image patching and Secrets

- Approach: patching the Pod image to deploy encryption proxy and custom networking
- Access to Docker daemon and images possible
  1. Re-tag original Pod image
  2. Use original image as base for patched version
  3. Build patched image with Tencrypt scripts and proxy app, tagged as „original“
- As mentioned, Secrets are either in ENV or mounts.
  Blocker: „Pod“ container has no access.

Docker image patching and Secrets

- Approach: patching the Pod image to deploy encryption proxy and custom networking
- Access to Docker daemon and images possible
  1. Re-tag original Pod image
  2. Use original image as base for patched version
  3. Build patched image with Tencrypt scripts and proxy app, tagged as „original“
- As mentioned, Secrets are either in ENV or mounts.
  Blocker: „Pod“ container has no access.

Part 1: Setting up the Pods network

- Traffic should either flow untouched or encrypted
- `eth0` adapter only egress interface in Pod
- Approach: introduce `tenc0` interface which has listening proxy application
- Configure network to route Service traffic through `tenc0` TUN
- Proxy application reads from interface, encrypts and forwards
- Blockers:
  - Pod has no capability, cannot self-configure network
  - Using manipulating Docker or Security Context Constraints (SCC) does not work
- Tencrypt: runs proxy from Host inside network namespace

Part 1: Setting up the Pods network

- Traffic should either flow untouched or encrypted
- `eth0` adapter only egress interface in Pod
- Approach: introduce `tenc0` interface which has listening proxy application
- Configure network to route Service traffic through `tenc0` TUN
- Proxy application reads from interface, encrypts and forwards
- Blockers:
  - Pod has no capability, cannot self-configure network
  - Using , manipulating Docker or Security Context Constraint (SCC) does not work
- Tencrypt: runs proxy from Host inside network namespace

Part 1: Setting up the Pods network

- Traffic should either flow untouched or encrypted
- `eth0` adapter only egress interface in Pod
- Approach: introduce `tenc0` interface which has listening proxy application
- Configure network to route Service traffic through `tenc0` TUN
- Proxy application reads from interface, encrypts and forwards
- Blockers:
  - Pod has no capability, cannot self-configure network
  - Using , manipulating Docker or Security Context Constraints (SCC) does not work
- Tencrypt: runs proxy from Host inside network namespace

Part 1: Setting up the Pods network

- Traffic should either flow untouched or encrypted
- `eth0` adapter only egress interface in Pod
- Approach: introduce `tenc0` interface which has listening proxy application
- Configure network to route Service traffic through `tenc0` TUN
- Proxy application reads from interface, encrypts and forwards
- Blockers:
  - Pod has no capability, cannot self-configure network
  - Using , manipulating Docker or Security Context Constraint (SCC) does not work
- Tencrypt: runs proxy from Host inside network namespace

Part 1: Setting up the Pods network

- Traffic should either flow untouched or encrypted
- `eth0` adapter only egress interface in Pod
- Approach: introduce `tenc0` interface which has listening proxy application
- Configure network to route Service traffic through `tenc0` TUN
- Proxy application reads from interface, encrypts and forwards
- Blockers:
  - Pod has no capability, cannot self-configure network
  - Using , manipulating Docker or Security Context Constraint (SCC) does not work
- Tencrypt: runs proxy from Host inside network namespace

Part 1: Setting up the Pods network

- Traffic should either flow untouched or encrypted
- `eth0` adapter only egress interface in Pod
- Approach: introduce `tenc0` interface which has listening proxy application
- Configure network to route Service traffic through `tenc0` TUN
- Proxy application reads from interface, encrypts and forwards
- Blockers:
  - Pod has no `NET_ADMIN` capability, cannot self-configure network
  - Using `setcap`, manipulating Docker or Security Context Constraint (SCC) does not work
- Tencrypt: runs proxy from Host inside network namespace

Part 1: Setting up the Pods network

- Traffic should either flow untouched or encrypted
- `eth0` adapter only egress interface in Pod
- Approach: introduce `tenc0` interface which has listening proxy application
- Configure network to route Service traffic through `tenc0` TUN
- Proxy application reads from interface, encrypts and forwards
- Blockers:
  - Pod has no `NET_ADMIN` capability, cannot self-configure network
  - Using `setcap`, manipulating Docker or Security Context Constraint (SCC) does not work
- Tencrypt: runs proxy from Host inside network namespace

Part 1: Setting up the Pods network

- Traffic should either flow untouched or encrypted
- `eth0` adapter only egress interface in Pod
- Approach: introduce `tenc0` interface which has listening proxy application
- Configure network to route Service traffic through `tenc0` TUN
- Proxy application reads from interface, encrypts and forwards
- Blockers:
  - Pod has no `NET_ADMIN` capability, cannot self-configure network
  - Using `setcap`, manipulating Docker or Security Context Constraint (SCC) does not work
- Tencrypt: runs proxy from Host inside network namespace

Part 1: Setting up the Pods network

- Traffic should either flow untouched or encrypted
- `eth0` adapter only egress interface in Pod
- Approach: introduce `tenc0` interface which has listening proxy application
- Configure network to route Service traffic through `tenc0` TUN
- Proxy application reads from interface, encrypts and forwards
- Blockers:
  - Pod has no `NET_ADMIN` capability, cannot self-configure network
  - Using `setcap`, manipulating Docker or Security Context Constraint (SCC) does not work
- Tencrypt: runs proxy from Host inside network namespace

Networking pitfalls

Part 2: Differentiation of Project-internal and -external traffic flows

- Traffic can be either Project-internal, -external or egress (NAT)
- Only Project-internal traffic should be encrypted
- Pods do not have access to `NAMESPACE` environment variable
- Services use DNS hierarchy with name of Project
- Approach: use DNS to identify type of remote (virtual) IP by reverse lookup
- Tencrypt: reads local `/etc/resolv.conf` for own name,
  queries hostnames of remote IPs and compares

Part 2: Differentiation of Project-internal and -external traffic flows

- Traffic can be either Project-internal, -external or egress (NAT)
- Only Project-internal traffic should be encrypted
- Pods do not have access to `NAMESPACE` environment variable
- Services use DNS hierarchy with name of Project
- Approach: use DNS to identify type of remote (virtual) IP by reverse lookup
- Tencrypt: reads local `/etc/resolv.conf` for own name,
  queries hostnames of remote IPs and compares

Part 2: Differentiation of Project-internal and -external traffic flows

- Traffic can be either Project-internal, -external or egress (NAT)
- Only Project-internal traffic should be encrypted
- Pods do not have access to `NAMESPACE` environment variable
- Services use DNS hierarchy with name of Project
- Approach: use DNS to identify type of remote (virtual) IP by reverse lookup
- Tencrypt: reads local `/etc/resolv.conf` for own name,
  queries hostnames of remote IPs and compares

Part 2: Differentiation of Project-internal and -external traffic flows

- Traffic can be either Project-internal, -external or egress (NAT)
- Only Project-internal traffic should be encrypted
- Pods do not have access to `NAMESPACE` environment variable
- Services use DNS hierarchy with name of Project
- Approach: use DNS to identify type of remote (virtual) IP by reverse lookup
- Tencrypt: reads local `/etc/resolv.conf` for own name, queries hostnames of remote IPs and compares

Part 2: Differentiation of Project-internal and -external traffic flows

- Traffic can be either Project-internal, -external or egress (NAT)
- Only Project-internal traffic should be encrypted
- Pods do not have access to `NAMESPACE` environment variable
- Services use DNS hierarchy with name of Project
- Approach: use DNS to identify type of remote (virtual) IP by reverse lookup
- Tencrypt: reads local `/etc/resolv.conf` for own name, queries hostnames of remote IPs and compares

Part 2: Differentiation of Project-internal and -external traffic flows

- Traffic can be either Project-internal, -external or egress (NAT)
- Only Project-internal traffic should be encrypted
- Pods do not have access to `NAMESPACE` environment variable
- Services use DNS hierarchy with name of Project
- Approach: use DNS to identify type of remote (virtual) IP by reverse lookup
- Tencrypt: reads local `/etc/resolv.conf` for own name,
  queries hostnames of remote IPs and compares

Part 3: Encryption of traffic

- Client issues a request to a remote Service, asks DNS and connects to VIP
- VIP is routed through `tenc0` interface, proxy app reads packets
- Proxy app encrypts and encapsulates packet, sends it as UDP payload to remote Tencrypt endpoint
- Remote Tencrypt UDP socket decrypts packet, changes destination address, forwards to local Service
- Reply same way but in reverse

- *Other ideas in paper: `IP_TRANSPARENT`, SOCKS5, enforcing encrypted traffic*

Part 3: Encryption of traffic

- Client issues a request to a remote Service, asks DNS and connects to VIP
- VIP is routed through `tenc0` interface, proxy app reads packets
- Proxy app encrypts and encapsulates packet, sends it as UDP payload to remote Tencrypt endpoint
- Remote Tencrypt UDP socket decrypts packet, changes destination address, forwards to local Service
- Reply same way but in reverse

- *Other ideas in paper: `IP_TRANSPARENT`, SOCKS5, enforcing encrypted traffic*

Part 3: Encryption of traffic

- Client issues a request to a remote Service, asks DNS and connects to VIP
- VIP is routed through `tenc0` interface, proxy app reads packets
- Proxy app encrypts and encapsulates packet, sends it as UDP payload to remote Tencrypt endpoint
- Remote Tencrypt UDP socket decrypts packet, changes destination address, forwards to local Service
- Reply same way but in reverse

- *Other ideas in paper: `IP_TRANSPARENT`, SOCKS5, enforcing encrypted traffic*

Part 3: Encryption of traffic

- Client issues a request to a remote Service, asks DNS and connects to VIP
- VIP is routed through `tenc0` interface, proxy app reads packets
- Proxy app encrypts and encapsulates packet, sends it as UDP payload to remote Tencrypt endpoint
- Remote Tencrypt UDP socket decrypts packet, changes destination address, forwards to local Service
- Reply same way but in reverse

- *Other ideas in paper: `IP_TRANSPARENT`, SOCKS5, enforcing encrypted traffic*

Part 3: Encryption of traffic

- Client issues a request to a remote Service, asks DNS and connects to VIP
- VIP is routed through `tenc0` interface, proxy app reads packets
- Proxy app encrypts and encapsulates packet, sends it as UDP payload to remote Tencrypt endpoint
- Remote Tencrypt UDP socket decrypts packet, changes destination address, forwards to local Service
- Reply same way but in reverse

- *Other ideas in paper: `IP_TRANSPARENT`, SOCKS5, enforcing encrypted traffic*

Part 3: Encryption of traffic

- Client issues a request to a remote Service, asks DNS and connects to VIP
- VIP is routed through `tenc0` interface, proxy app reads packets
- Proxy app encrypts and encapsulates packet, sends it as UDP payload to remote Tencrypt endpoint
- Remote Tencrypt UDP socket decrypts packet, changes destination address, forwards to local Service
- Reply same way but in reverse

- *Other ideas in paper: `IP_TRANSPARENT`, SOCKS5, enforcing encrypted traffic*

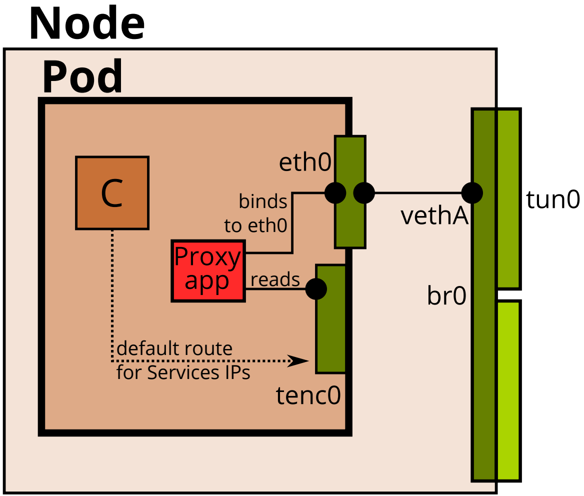

Figure 2: Overview of the proxy application implementation schema

Proof of concept implementation

- Four components:
  1. **DNS upstream proxy**, parsing replies
  2. **TUN interface handler**, reading packets and writing replies
  3. **UDP encapsulation**, encryption and decryption, UDP **listener** on a specified port
  4. **Raw sockets** to forward packets on local interface
- DNS proxy uses static connection to upstream
- White-listing of external hosts
- DNAT on received packets

Flow for DNS request/reply

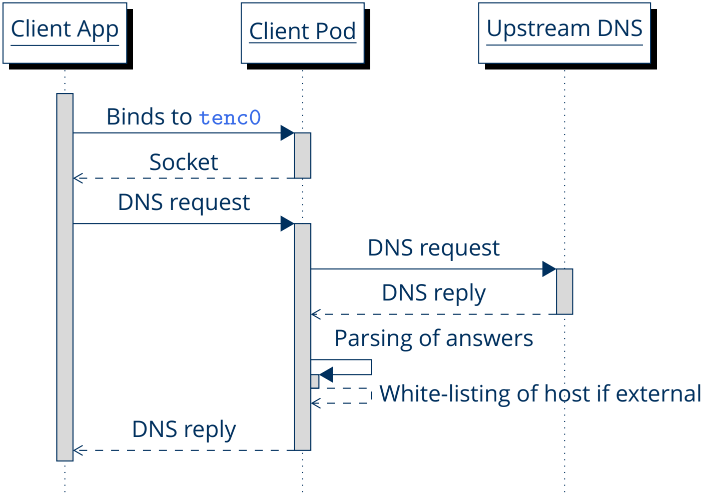

Figure 3: Traffic flow part one, DNS proxy

Flow for client request to remote Service

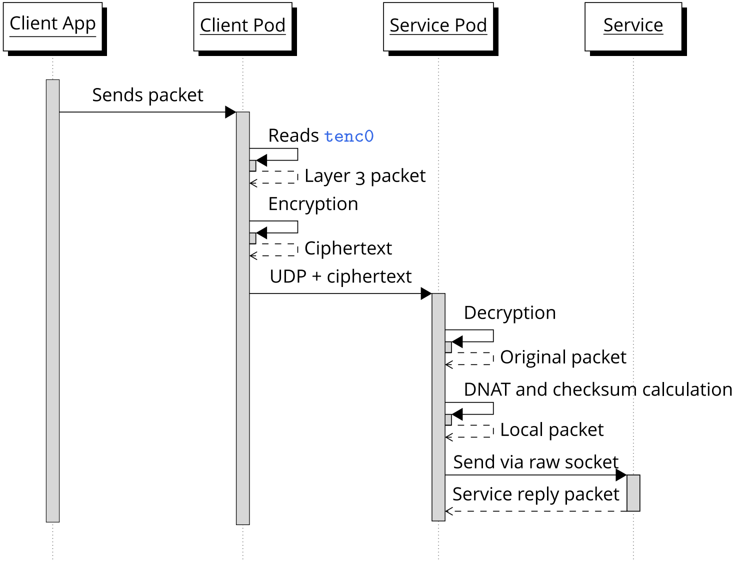

Figure 4: Traffic flow part two, client issues a request to a Service

Flow for Service replies

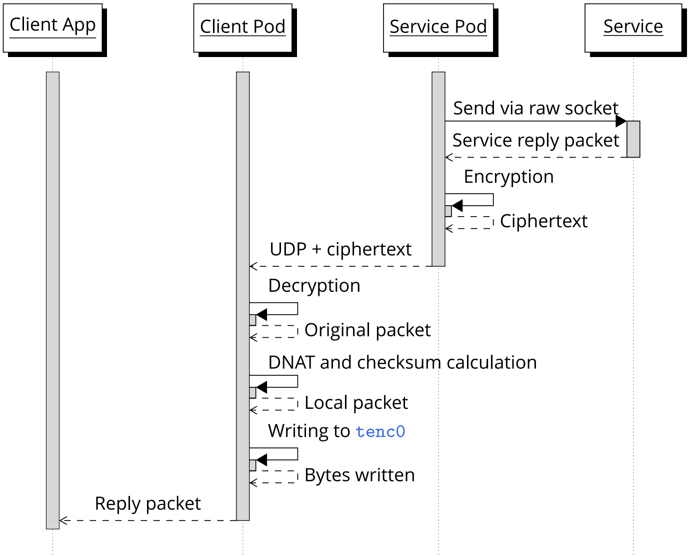

Figure 5: Traffic flow part three, Service replies to request

Throughput measurements

Test description	Transmitted	Throughput server	Throughput client
no patch, 1 client	22640.38 MB	2262.3 MB/s	2263.83 MB/s
no patch, 5 clients	27350.0 MB	2724.02 MB/s	2727.22 MB/s
no patch, 10 clients	25817.38 MB	2410.15 MB/s	2538.02 MB/s
patched, 1 client	2.46 MB	0.01 MB/s	0.16 MB/s
patched, 5 clients	12.29 MB	0.06 MB/s	0.82 MB/s
patched, 10 clients	24.58 MB	0.13 MB/s	1.64 MB/s
patched, 20 clients	24.86 MB	0.12 MB/s	3.27 MB/s

Figure 6: Results of iperf measurements with different amounts of clients. Taken with iperf inside the Minishift VM (Virtualbox, 2 CPUs, 2GB RAM).

Observation: amount of clients in patched environment makes a difference, bandwidth is summed up. Probable cause: caching. Should not be taken too seriously, as implementation is unoptimised proof of concept.

Conclusion

- Original aim: transparent encryption of Pod-to-Pod traffic per Tenant
- Step 1: Analysis of the OpenShift network setup
- Step 2: Defining security requirements and threats
- Step 3: Design approach alternatives
- Step 4: Exploration of different implementation parts
- Step 5: Proof of concept implementation
- Result: implementation works, concept proved to be usable
- Future work: test integration of concept into existing OpenShift code base, re-write proxy application

Conclusion

- Original aim: transparent encryption of Pod-to-Pod traffic per Tenant
- Step 1: Analysis of the OpenShift network setup
- Step 2: Defining security requirements and threats
- Step 3: Design approach alternatives
- Step 4: Exploration of different implementation parts
- Step 5: Proof of concept implementation
- Result: implementation works, concept proved to be usable
- Future work: test integration of concept into existing OpenShift code base, re-write proxy application

Conclusion

- Original aim: transparent encryption of Pod-to-Pod traffic per Tenant
- Step 1: Analysis of the OpenShift network setup
- Step 2: Defining security requirements and threats
- Step 3: Design approach alternatives
- Step 4: Exploration of different implementation parts
- Step 5: Proof of concept implementation
- Result: implementation works, concept proved to be usable
- Future work: test integration of concept into existing OpenShift code base, re-write proxy application

Conclusion

- Original aim: transparent encryption of Pod-to-Pod traffic per Tenant
- Step 1: Analysis of the OpenShift network setup
- Step 2: Defining security requirements and threats
- Step 3: Design approach alternatives
- Step 4: Exploration of different implementation parts
- Step 5: Proof of concept implementation
- Result: implementation works, concept proved to be usable
- Future work: test integration of concept into existing OpenShift code base, re-write proxy application

Conclusion

- Original aim: transparent encryption of Pod-to-Pod traffic per Tenant
- Step 1: Analysis of the OpenShift network setup
- Step 2: Defining security requirements and threats
- Step 3: Design approach alternatives
- Step 4: Exploration of different implementation parts
- Step 5: Proof of concept implementation
- Result: implementation works, concept proved to be usable
- Future work: test integration of concept into existing OpenShift code base, re-write proxy application

Conclusion

- Original aim: transparent encryption of Pod-to-Pod traffic per Tenant
- Step 1: Analysis of the OpenShift network setup
- Step 2: Defining security requirements and threats
- Step 3: Design approach alternatives
- Step 4: Exploration of different implementation parts
- Step 5: Proof of concept implementation
- Result: implementation works, concept proved to be usable
- Future work: test integration of concept into existing OpenShift code base, re-write proxy application

Conclusion

- Original aim: transparent encryption of Pod-to-Pod traffic per Tenant
- Step 1: Analysis of the OpenShift network setup
- Step 2: Defining security requirements and threats
- Step 3: Design approach alternatives
- Step 4: Exploration of different implementation parts
- Step 5: Proof of concept implementation
- Result: implementation works, concept proved to be usable
- Future work: test integration of concept into existing OpenShift code base, re-write proxy application

Conclusion

- Original aim: transparent encryption of Pod-to-Pod traffic per Tenant
- Step 1: Analysis of the OpenShift network setup
- Step 2: Defining security requirements and threats
- Step 3: Design approach alternatives
- Step 4: Exploration of different implementation parts
- Step 5: Proof of concept implementation
- Result: implementation works, concept proved to be usable
- Future work: test integration of concept into existing OpenShift code base, re-write proxy application

Sources and further information

These slides and the associated report with further references will be published on my website `https://dpataky.eu` and can be used under the CC BY-SA 4.0 license.

[Sho14] Adam Shostack. *Threat modeling designing for security*. J. Wiley & Sons, 2014. ISBN: 9781118809990.

Appendix

Note about Containerisation

- Threat analysis does not cover attacks on a host by a priviledged attacker
- Administrative access to namespaces cannot be fend off by Tencrypt
- Future mechanisms might solve this problem (hardware-based memory isolation)
- But: Tencrypt stays inside namespace and generally reduces attack vectors

Note about Containerisation

- Threat analysis does not cover attacks on a host by a priviledged attacker
- Administrative access to namespaces cannot be fend off by Tencrypt
- Future mechanisms might solve this problem
(hardware-based memory isolation)
- But: Tencrypt stays inside namespace and generally reduces attack vectors

Note about Containerisation

- Threat analysis does not cover attacks on a host by a priviledged attacker
- Administrative access to namespaces cannot be fend off by Tencrypt
- Future mechanisms might solve this problem (hardware-based memory isolation)
- But: Tencrypt stays inside namespace and generally reduces attack vectors

Note about Containerisation

- Threat analysis does not cover attacks on a host by a priviledged attacker
- Administrative access to namespaces cannot be fend off by Tencrypt
- Future mechanisms might solve this problem (hardware-based memory isolation)
- But: Tencrypt stays inside namespace and generally reduces attack vectors

Pitfalls in networking

- Boot2Docker misses some features, `nsenter` only in CentOS
- Approach to use local proxy application with DNAT fails, because addresses are lost (combination of IPtables `mangle` and `TPROXY`)
- Using TAP interface on layer 2 requires implementation of ARP
- Routes might result in loop, proxy is routed through itself
- Tencrypt: uses policy-based routing (`ip rule`), two tables and 1440 bytes MTU

Pitfalls in networking

- Boot2Docker misses some features, `nsenter` only in CentOS
- Approach to use local proxy application with DNAT fails, because addresses are lost (combination of IPtables `mangle` and `TPROXY`)
- Using TAP interface on layer 2 requires implementation of ARP
- Routes might result in loop, proxy is routed through itself
- Tencrypt: uses policy-based routing (`ip rule`), two tables and 1440 bytes MTU

Pitfalls in networking

- Boot2Docker misses some features, `nsenter` only in CentOS
- Approach to use local proxy application with DNAT fails, because addresses are lost (combination of IPtables `mangle` and `TPROXY`)
- Using TAP interface on layer 2 requires implementation of ARP
- Routes might result in loop, proxy is routed through itself
- Tencrypt: uses policy-based routing (`ip rule`), two tables and 1440 bytes MTU

Pitfalls in networking

- Boot2Docker misses some features, `nsenter` only in CentOS
- Approach to use local proxy application with DNAT fails, because addresses are lost (combination of IPtables `mangle` and `TPROXY`)
- Using TAP interface on layer 2 requires implementation of ARP
- Routes might result in loop, proxy is routed through itself
- Tencrypt: uses policy-based routing (`ip rule`), two tables and 1440 bytes MTU

Pitfalls in networking

- Boot2Docker misses some features, `nsenter` only in CentOS
- Approach to use local proxy application with DNAT fails, because addresses are lost (combination of IPtables `mangle` and `TPROXY`)
- Using TAP interface on layer 2 requires implementation of ARP
- Routes might result in loop, proxy is routed through itself
- Tencrypt: uses policy-based routing (`ip rule`), two tables and 1440 bytes MTU

Related tools and products

- **OKD Secured Routes**: encryption of traffic from ingress Router to Service
- **IPsec**: encrypted Node-to-Node and Node-to-Master channels

- **Aporeto**: security suite for Kubernetes in different Cloud environments. Offers TLS-encrypted end-to-end encryption and policies.
- **Aqua Security**: Container security by using Container-level firewalls, but no encryption
- **Istio Auth**: uses „Envoy“ Service proxies inside Pods to tunnel traffic with mTLS. Uses the Secure Production Identity Framework for Everyone (SPIFFE) framework.

Related tools and products

- **OKD Secured Routes**: encryption of traffic from ingress Router to Service
- **IPsec**: encrypted Node-to-Node and Node-to-Master channels

- **Aporeto**: security suite for Kubernetes in different Cloud environments. Offers TLS-encrypted end-to-end encryption and policies.
- **Aqua Security**: Container security by using Container-level firewalls, but no encryption
- **Istio Auth**: uses „Envoy" Service proxies inside Pods to tunnel traffic with mTLS. Uses the Secure Production Identity Framework for Everyone (SPIFFE) framework.

Related tools and products

- **OKD Secured Routes**: encryption of traffic from ingress Router to Service
- **IPsec**: encrypted Node-to-Node and Node-to-Master channels

---

- **Aporeto**: security suite for Kubernetes in different Cloud environments. Offers TLS-encrypted end-to-end encryption and policies.
- **Aqua Security**: Container security by using Container-level firewalls, but no encryption
- **Istio Auth**: uses „Envoy" Service proxies inside Pods to tunnel traffic with mTLS. Uses the Secure Production Identity Framework for Everyone (SPIFFE) framework.

Related tools and products

- **OKD Secured Routes**: encryption of traffic from ingress Router to Service
- **IPsec**: encrypted Node-to-Node and Node-to-Master channels

---

- **Aporeto**: security suite for Kubernetes in different Cloud environments. Offers TLS-encrypted end-to-end encryption and policies.
- **Aqua Security**: Container security by using Container-level firewalls, but no encryption
- **Istio Auth**: uses „Envoy" Service proxies inside Pods to tunnel traffic with mTLS. Uses the Secure Production Identity Framework for Everyone (SPIFFE) framework.

Related tools and products

- **OKD Secured Routes**: encryption of traffic from ingress Router to Service
- **IPsec**: encrypted Node-to-Node and Node-to-Master channels

---

- **Aporeto**: security suite for Kubernetes in different Cloud environments. Offers TLS-encrypted end-to-end encryption and policies.
- **Aqua Security**: Container security by using Container-level firewalls, but no encryption
- **Istio Auth**: uses „Envoy“ Service proxies inside Pods to tunnel traffic with mTLS. Uses the Secure Production Identity Framework for Everyone (SPIFFE) framework.

Related technologies

- **MACsec**: encryption and integrity on layer 2. Extends Ethernet frames with MACsec tag. Standard as IEEE 802.1AE.
- **Single Root I/O Virtualization (SR-IOV)**: virtualisation of PCI Express hardware resources done by the board itself. VMs see shared PCI-E components as exclusive.
- **Cisco Application Centric Infrastructure (ACI)**: SDN-oriented policy-based framework developed by Cisco. Better management with VXLAN, ECMP routing and SDN controllers.

Related technologies

- **MACsec**: encryption and integrity on layer 2. Extends Ethernet frames with MACsec tag. Standard as IEEE 802.1AE.
- **Single Root I/O Virtualization (SR-IOV)**: virtualisation of PCI Express hardware resources done by the board itself. VMs see shared PCI-E components as exclusive.
- **Cisco Application Centric Infrastructure (ACI)**: SDN-oriented policy-based framework developed by Cisco. Better management with VXLAN, ECMP routing and SDN controllers.

Related technologies

- **MACsec**: encryption and integrity on layer 2. Extends Ethernet frames with MACsec tag. Standard as IEEE 802.1AE.
- **Single Root I/O Virtualization (SR-IOV)**: virtualisation of PCI Express hardware resources done by the board itself. VMs see shared PCI-E components as exclusive.
- **Cisco Application Centric Infrastructure (ACI)**: SDN-oriented policy-based framework developed by Cisco. Better management with VXLAN, ECMP routing and SDN controllers.