TECHNISCHE UNIVERSITÄT DRESDEN

Dominik Pataky

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

SecShift: Analysis and Conception of Traffic Security for the OpenShift Platform

Diplomarbeit // Dresden, 4th July, 2019

Context

- So far: companies use in-house hardware according to their needs

- Problem: increased need for temporary computing resources
- Solution: the 'Cloud', on demand usage of centralised resources

- Problem: security, separation of tenants in shared environments
- Solution: virtualisation and containerisation

- Problem: data is routed through foreign infrastructure
- Solution: network traffic security

- Problem: cool, but how?
- Solution: SecShift!

Context

- So far: companies use in-house hardware according to their needs
- Problem: increased need for temporary computing resources
- Solution: the 'Cloud', on demand usage of centralised resources
- Problem: security, separation of tenants in shared environments
- Solution: virtualisation and containerisation
- Problem: data is routed through foreign infrastructure
- Solution: network traffic security
- Problem: cool, but how?
- Solution: SecShift!

Context

- So far: companies use in-house hardware according to their needs
- Problem: increased need for temporary computing resources
- Solution: the 'Cloud', on demand usage of centralised resources
- Problem: security, separation of tenants in shared environments
- Solution: virtualisation and containerisation
- Problem: data is routed through foreign infrastructure
- Solution: network traffic security
- Problem: cool, but how?
- Solution: SecShift!

Context

- So far: companies use in-house hardware according to their needs
- Problem: increased need for temporary computing resources
- Solution: the ’Cloud’, on demand usage of centralised resources
- Problem: security, separation of tenants in shared environments
- Solution: virtualisation and containerisation
- Problem: data is routed through foreign infrastructure
- Solution: network traffic security
- Problem: cool, but how?
- Solution: SecShift!

Context

- So far: companies use in-house hardware according to their needs
- Problem: increased need for temporary computing resources
- Solution: the 'Cloud', on demand usage of centralised resources
- Problem: security, separation of tenants in shared environments
- Solution: virtualisation and containerisation
- Problem: data is routed through foreign infrastructure
- Solution: network traffic security
- Problem: cool, but how?
- Solution: SecShift!

The seven steps toward SecShift

1. What's the technology stack?
2. Scope of the problem domain?
3. Identifiable threats in the topology? Additional requirements?
4. Existing work and products?
5. New design – variations?
6. Implementable in practice?
7. Does it work? Performance? Requirements fulfilled?

The seven steps toward SecShift

1. What's the technology stack?
2. Scope of the problem domain?
3. Identifiable threats in the topology? Additional requirements?
4. Existing work and products?
5. New design – variations?
6. Implementable in practice?
7. Does it work? Performance? Requirements fulfilled?

The seven steps toward SecShift

1. What's the technology stack?
2. Scope of the problem domain?
3. Identifiable threats in the topology? Additional requirements?
4. Existing work and products?
5. New design – variations?
6. Implementable in practice?
7. Does it work? Performance? Requirements fulfilled?

The seven steps toward SecShift

1. What's the technology stack?
2. Scope of the problem domain?
3. Identifiable threats in the topology? Additional requirements?
4. Existing work and products?
5. New design – variations?
6. Implementable in practice?
7. Does it work? Performance? Requirements fulfilled?

The seven steps toward SecShift

1. What's the technology stack?
2. Scope of the problem domain?
3. Identifiable threats in the topology? Additional requirements?
4. Existing work and products?
5. New design – variations?
6. Implementable in practice?
7. Does it work? Performance? Requirements fulfilled?

The seven steps toward SecShift

1. What's the technology stack?
2. Scope of the problem domain?
3. Identifiable threats in the topology? Additional requirements?
4. Existing work and products?
5. New design – variations?
6. Implementable in practice?
7. Does it work? Performance? Requirements fulfilled?

The seven steps toward SecShift

1. What's the technology stack?
2. Scope of the problem domain?
3. Identifiable threats in the topology? Additional requirements?
4. Existing work and products?
5. New design – variations?
6. Implementable in practice?
7. Does it work? Performance? Requirements fulfilled?

The seven steps toward SecShift

1. What's the technology stack?
2. Scope of the problem domain?
3. Identifiable threats in the topology? Additional requirements?
4. Existing work and products?
5. New design – variations?
6. Implementable in practice?
7. Does it work? Performance? Requirements fulfilled?

But first.. OpenShift!

Look into the OpenShift cloud platform

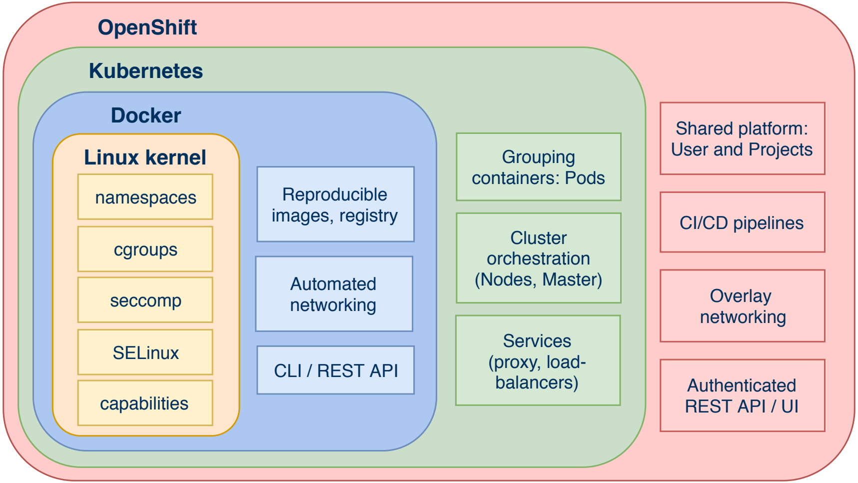

Figure 1: Overview over components in OpenShift

Essential components of OpenShift

Let's be a user and deploy a web and database app combination!

1. We define two **container** images (isolated applications)
2. OpenShift creates two **pods** in our Project
3. The **Master** schedules the Pods on **Nodes** (Linux machines).
4. Connection through **service layer** (load-balancing, virtual IPs)

Essential components of OpenShift

Let's be a user and deploy a web and database app combination!

1. We define two **container** images (isolated applications)
2. OpenShift creates two **pods** in our Project
3. The **Master** schedules the Pods on **Nodes** (Linux machines).
4. Connection through **service layer** (load-balancing, virtual IPs)

Essential components of OpenShift

Let's be a user and deploy a web and database app combination!

1. We define two **container** images (isolated applications)
2. OpenShift creates two **pods** in our Project
3. The **Master** schedules the Pods on **Nodes** (Linux machines).
4. Connection through **service layer** (load-balancing, virtual IPs)

Essential components of OpenShift

Let's be a user and deploy a web and database app combination!

1. We define two **container** images (isolated applications)
2. OpenShift creates two **pods** in our Project
3. The **Master** schedules the Pods on **Nodes** (Linux machines).
4. Connection through **service layer** (load-balancing, virtual IPs)

Essential components of OpenShift

Let's be a user and deploy a web and database app combination!

1. We define two **container** images (isolated applications)
2. OpenShift creates two **pods** in our Project
3. The **Master** schedules the Pods on **Nodes** (Linux machines).
4. Connection through **service layer** (load-balancing, virtual IPs)

Upcoming task: secure connection between webserver and database!

OpenShift networking | OpenShift security

Problem domain model

1. Base: OpenShift topology
2. Set preconditions to limit scope
3. Definition of entities (components and interconnections), traffic flows and adversary models
4. Refinement of scope: focus on traffic security
5. Threat modelling in new scope

Problem domain model

1. Base: OpenShift topology
2. Set preconditions to limit scope
3. Definition of entities (components and interconnections), traffic flows and adversary models
4. Refinement of scope: focus on traffic security
5. Threat modelling in new scope

Problem domain model

1. Base: OpenShift topology
2. Set preconditions to limit scope
3. Definition of entities (components and interconnections), traffic flows and adversary models
4. Refinement of scope: focus on traffic security
5. Threat modelling in new scope

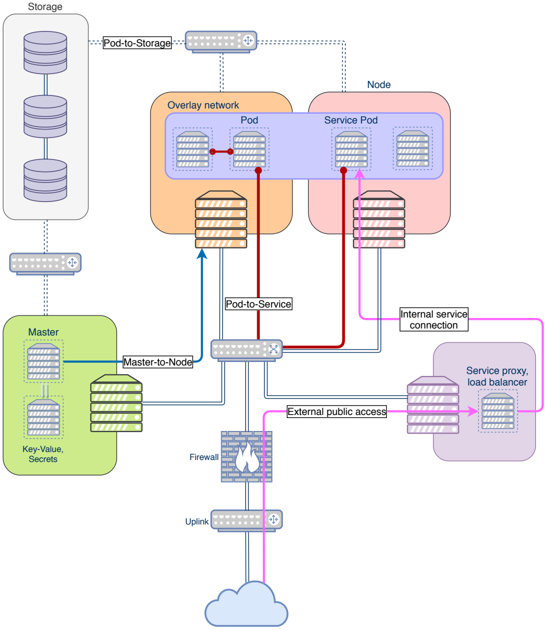

Figure 2: Problem domain topology

Problem domain model

1. Base: OpenShift topology
2. Set preconditions to limit scope
3. Definition of entities (components and interconnections), traffic flows and adversary models
4. Refinement of scope: focus on traffic security
5. Threat modelling in new scope

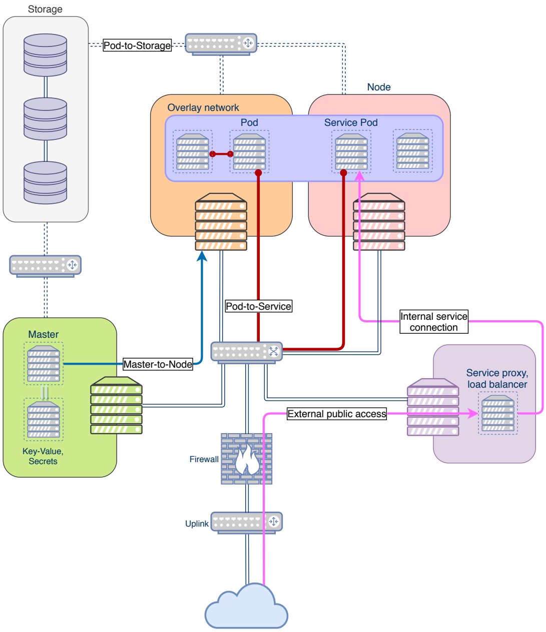

Figure 2: Problem domain topology

Problem domain model

1. Base: OpenShift topology
2. Set preconditions to limit scope
3. Definition of entities (components and interconnections), traffic flows and adversary models
4. Refinement of scope: focus on traffic security
5. Threat modelling in new scope

Traffic flows and adversaries

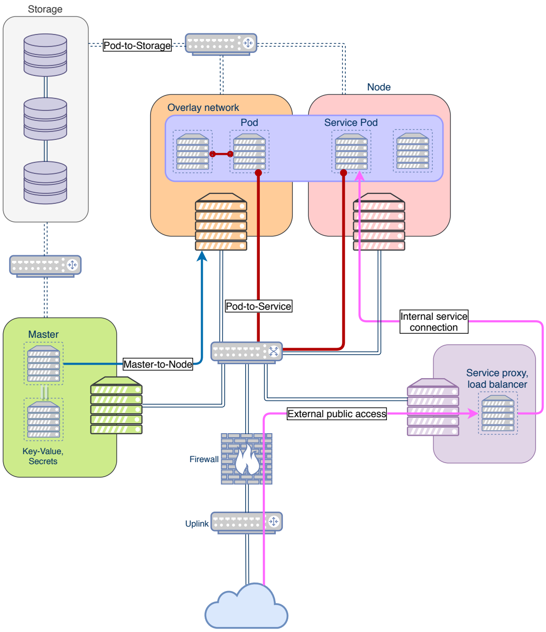

Figure 2: Problem domain topology

Threats and requirements analysis

1. STRIDE: threat modelling

   Amount of threats $\rightarrow$ ranking $\odot \oslash \otimes$

Threat	Protection goal
Spoofing	Authenticity
Tampering	Integrity
Repudiability	Non-repudiability
Info disclosure	Confidentiality
Denial of Service	Availability
Elevation of privilege	Authorisation

2. Security requirements for traffic security and key management

   based on STRIDE, $2 * 6 = 12$ goals

3. SQuaRE: 'Systems and software Quality Requirements and Evaluation' (ISO standard)

   10 characteristics

Threats and requirements analysis

1. STRIDE: threat modelling

   Amount of threats $\rightarrow$ ranking $\odot \oslash \otimes$

Threat	Protection goal
Spoofing	Authenticity
Tampering	Integrity
Repudiability	Non-repudiability
Info disclosure	Confidentiality
Denial of Service	Availability
Elevation of privilege	Authorisation

2. Security requirements for traffic security and key management

   based on STRIDE, $2 * 6 = 12$ goals

3. SQuaRE: ‘Systems and software Quality Requirements and Evaluation’ (ISO standard)

   10 characteristics

Threats and requirements analysis

1. STRIDE: threat modelling

   Amount of threats → ranking ⊙⊘⊗

Threat	Protection goal
Spoofing	Authenticity
Tampering	Integrity
Repudiability	Non-repudiability
Info disclosure	Confidentiality
Denial of Service	Availability
Elevation of privilege	Authorisation

2. Security requirements for traffic security and key management

   based on STRIDE, $2 * 6 = 12$ goals

3. SQuaRE: 'Systems and software Quality Requirements and Evaluation' (ISO standard)

   10 characteristics

Threats and requirements analysis

1. STRIDE: threat modelling

   Amount of threats → ranking ⊙⊘⊗

Threat	Protection goal
Spoofing	Authenticity
Tampering	Integrity
Repudiability	Non-repudiability
Info disclosure	Confidentiality
Denial of Service	Availability
Elevation of privilege	Authorisation

2. Security requirements for traffic security and key management

   based on STRIDE, $2 * 6 = 12$ goals

3. SQuaRE: 'Systems and software Quality Requirements and Evaluation' (ISO standard)

   10 characteristics

Now we know what must be covered.

What are existing threats?

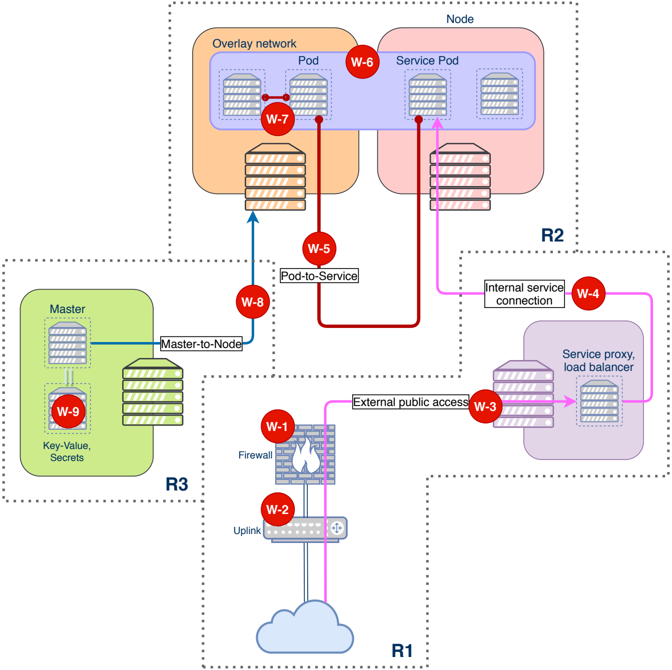

Figure 3: OpenShift topology with weak points (weaknesses W-1 to W-9) and responsibility groups R1-R3

Threat analysis for entities

RG	ID	Description	STRIDE
R1	W-1	The firewall, security layer dividing security zones.	STRIDE ⊗
	W-2	The uplink, connecting data centers to the public internet.	STRID- ⊗
	W-3	Traffic flowing through the external public access.	-T-ID- ⊘
	W-4	Traffic forwarded by the Service Proxy onto the internal Service connection.	-TRID- ⊘
R2	W-5	Pod-to-Service (P2S) connections from Pods to virtual IPs.	STRIDE ⊗
	W-6	Connections routed via the overlay network (ON).	STRIDE ⊗
	W-7	Pod-to-Pod (P2P) traffic on the same Node.	-T-IDE ⊘
R3	W-8	The bidirectional Master-to-Node (M2N) connections.	S--IDE ⊘
	W-9	Access to the Secrets database (SDB).	ST-ID- ⊘

Table 1: Weaknesses of components and interconnections in network related contexts. R*n* references the responsibility group. W-*n* reference the identifiers in fig. 3. Symbols are ranked as low (⊙), mid (⊘) and high (⊗) criticality according to their STRIDE vulnerability.

Threat analysis for entities

RG	ID	Description	STRIDE
R1	W-1	The firewall, security layer dividing security zones.	STRIDE ⊗
	W-2	The uplink, connecting data centers to the public internet.	STRID- ⊗
	W-3	Traffic flowing through the external public access.	-T-ID- ⊘
	W-4	Traffic forwarded by the Service Proxy onto the internal Service connection.	-TRID- ⊘
R2	W-5	Pod-to-Service (P2S) connections from Pods to virtual IPs.	STRIDE ⊗
	W-6	Connections routed via the overlay network (ON).	STRIDE ⊗
	W-7	Pod-to-Pod (P2P) traffic on the same Node.	-T-IDE ⊘
R3	W-8	The bidirectional Master-to-Node (M2N) connections.	S--IDE ⊘
	W-9	Access to the Secrets database (SDB).	ST-ID- ⊘

Table 1: Weaknesses of components and interconnections in network related contexts. R*n* references the responsibility group. W-*n* reference the identifiers in fig. 3. Symbols are ranked as low (⊙), mid (⊘) and high (⊗) criticality according to their STRIDE vulnerability.

Short break – we now know..

✓ How OpenShift works

✓ The problem domain for SecShift

✓ Which threats exist, with focus on Pod-to-Pod traffic security

✓ Goals beyond threat mitigation

Next: existing work and technology, followed by SecShift's design

What exists?

- Predecessor '**Tencrypt**: hardening OpenShift by encrypting tenant traffic'
- Research: tcpcrypt, multi-tenancy in the cloud, performance analysis of VPN software, security examinations, **Zero Trust networking**
- Technology: Kubernetes extensions Istio **Envoy proxies**, Wormhole and Cilium encryption layer, VPN software, secrets management
- Related: **memory protection** (secure enclaves), improved isolation of applications with Kata Containers and Firecracker, layer 2 encryption with MACsec (IEEE standard)
- **SecShift**: integrated encryption overlay for each tenant, no application adaptation through transparent routing, utilising namespaces, distributed architecture

What exists?

- Predecessor '**Tencrypt**: hardening OpenShift by encrypting tenant traffic'
- Research: tcpcrypt, multi-tenancy in the cloud, performance analysis of VPN software, security examinations, **Zero Trust networking**
- Technology: Kubernetes extensions Istio **Envoy proxies**, Wormhole and Cilium encryption layer, VPN software, secrets management
- Related: **memory protection** (secure enclaves), improved isolation of applications with Kata Containers and Firecracker, layer 2 encryption with MACsec (IEEE standard)
- **SecShift**: integrated encryption overlay for each tenant, no application adaptation through transparent routing, utilising namespaces, distributed architecture

What exists?

- Predecessor '**Tencrypt**: hardening OpenShift by encrypting tenant traffic'
- Research: tcpcrypt, multi-tenancy in the cloud, performance analysis of VPN software, security examinations, **Zero Trust networking**
- Technology: Kubernetes extensions Istio **Envoy proxies**, Wormhole and Cilium encryption layer, VPN software, secrets management
- Related: **memory protection** (secure enclaves), improved isolation of applications with Kata Containers and Firecracker, layer 2 encryption with MACsec (IEEE standard)
- **SecShift**: integrated encryption overlay for each tenant, no application adaptation through transparent routing, utilising namespaces, distributed architecture

What exists?

- Predecessor '**Tencrypt**: hardening OpenShift by encrypting tenant traffic'
- Research: tcpcrypt, multi-tenancy in the cloud, performance analysis of VPN software, security examinations, **Zero Trust networking**
- Technology: Kubernetes extensions Istio **Envoy proxies**, Wormhole and Cilium encryption layer, VPN software, secrets management
- Related: **memory protection** (secure enclaves), improved isolation of applications with Kata Containers and Firecracker, layer 2 encryption with MACsec (IEEE standard)
- **SecShift**: integrated encryption overlay for each tenant, no application adaptation through transparent routing, utilising namespaces, distributed architecture

What exists?

- Predecessor '**Tencrypt**: hardening OpenShift by encrypting tenant traffic'
- Research: tcpcrypt, multi-tenancy in the cloud, performance analysis of VPN software, security examinations, **Zero Trust networking**
- Technology: Kubernetes extensions Istio **Envoy proxies**, Wormhole and Cilium encryption layer, VPN software, secrets management
- Related: **memory protection** (secure enclaves), improved isolation of applications with Kata Containers and Firecracker, layer 2 encryption with MACsec (IEEE standard)

- **SecShift**: integrated encryption overlay for each tenant, no application adaptation through transparent routing, utilising namespaces, distributed architecture

SecShift design overview

- Multiple aspects (key type, peering, updates, . . .) taken into account
- Each design aspect has multiple possible approaches
- Two topology variations: hybrid and fully distributed
- Hybrid topology: three types of components used
  - SecShift Tenant Node daemon (STNd) and SecShift Pod daemon (SPd)
  - OpenShift Secret, Pods and Services APIs (centralised)
  - Container engine (here: Docker)

SecShift design overview

- Multiple aspects (key type, peering, updates, . . .) taken into account
- Each design aspect has multiple possible approaches
- Two topology variations: hybrid and fully distributed
- Hybrid topology: three types of components used
  - SecShift Tenant Node daemon (STNd) and SecShift Pod daemon (SPd)
  - OpenShift Secret, Pods and Services APIs *(centralised?)*
  - Container engine (here: Docker)

SecShift design overview

- Multiple aspects (key type, peering, updates, . . .) taken into account
- Each design aspect has multiple possible approaches
- Two topology variations: hybrid and fully distributed
- Hybrid topology: three types of components used
  - SecShift Tenant Node daemon (STNd) and SecShift Pod daemon (SPd)
  - OpenShift Secret, Pods and Services APIs (centralised)
  - Container engine (here: Docker)

SecShift design overview

- Multiple aspects (key type, peering, updates, . . .) taken into account
- Each design aspect has multiple possible approaches
- Two topology variations: hybrid and fully distributed
- Hybrid topology: three types of components used
  - SecShift Tenant Node daemon (STNd) and SecShift Pod daemon (SPd)
  - OpenShift Secret, Pods and Services APIs *(centralised!)*
  - Container engine (here: Docker)

SecShift design overview

- Multiple aspects (key type, peering, updates, . . .) taken into account
- Each design aspect has multiple possible approaches
- Two topology variations: hybrid and fully distributed
- Hybrid topology: three types of components used
  - SecShift Tenant Node daemon (STNd) and SecShift Pod daemon (SPd)
  - OpenShift Secret, Pods and Services APIs *(centralised!)*
  - Container engine (here: Docker)

SecShift design overview

- Multiple aspects (key type, peering, updates, . . .) taken into account
- Each design aspect has multiple possible approaches
- Two topology variations: hybrid and fully distributed
- Hybrid topology: three types of components used
  - SecShift Tenant Node daemon (STNd) and SecShift Pod daemon (SPd)
  - OpenShift Secret, Pods and Services APIs *(centralised!)*
  - Container engine (here: Docker)

Next: a look at the topology!

SecShift hybrid design

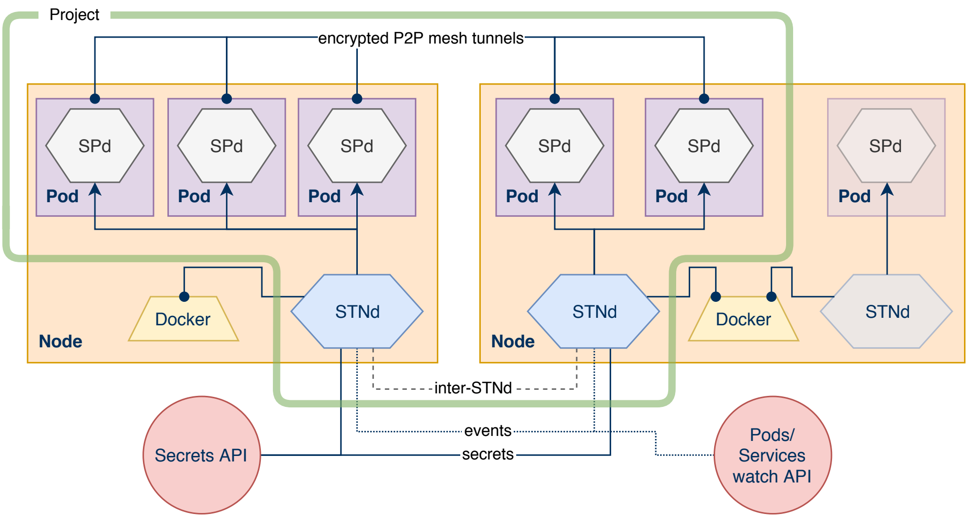

Figure 4: SecShift hybrid design topology

Evaluation!

Demo with functionality evaluation

The demo shows SecShift in action. At first the setup with Ansible is shown. Then, the vanilla OpenShift setup is used to demonstrate the ability for hosts to capture clear text traffic on the overlay network interface. Running SecShift and applying the encryption overlay then illustrates the changes: all Project-internal packets are routed transparently through tunnels in the meshed Pod-to-Pod network. Listening on the node's interfaces (VXLAN) visualises the encrypted packet stream.

Reference implementation

It works, but is it viable? A look at the performance. . .

Throughput for TCP in unsecured setup

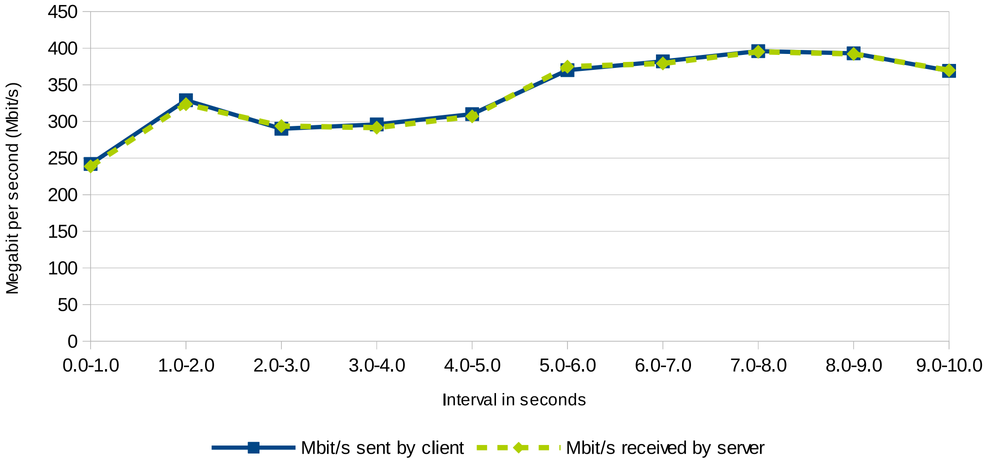

Figure 5: iperf with unsecured cross-Node connections, TCP, no bandwidth limit

Throughput in secured setup with TCP

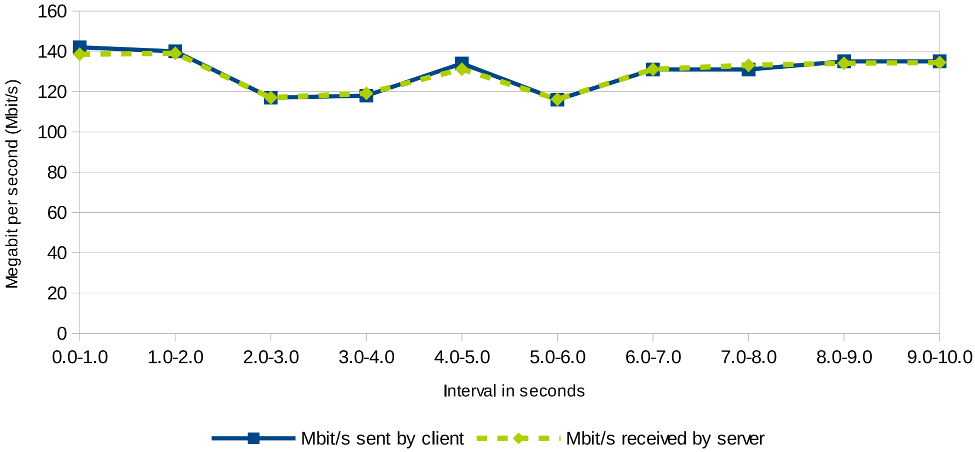

Figure 6: iperf with secure cross-Node tunnels, TCP, no bandwidth limit, no peer updates

It provides fair bandwidth!

Does it also meet the requirements?

Reviewing the requirements

- Weaknesses and STRIDE
  - Pod-to-Service (W-5): entirely bypassed $\otimes \rightarrow \odot$
  - Overlay network (W-6): key pairs, peer configuration and VPN features $\otimes \rightarrow \odot$
  - Pod-to-Pod on same Node (W-7): encryption in namespace $\oslash \rightarrow \odot$

- Security requirements: 10 of 12.

- SQuaRE characteristics: 8 of 10.

Reviewing the requirements

- Weaknesses and STRIDE
  - Pod-to-Service (W-5): entirely bypassed ⊗ → ⊙
  - Overlay network (W-6): key pairs, peer configuration and VPN features ⊗ → ⊙
  - Pod-to-Pod on same Node (W-7): encryption in namespace ⊘ → ⊙
- Security requirements: 10 of 12.
- SQuaRE characteristics: 8 of 10.

Reviewing the requirements

- Weaknesses and STRIDE
  - Pod-to-Service (W-5): entirely bypassed ⊗ → ⊙
  - Overlay network (W-6): key pairs, peer configuration and VPN features ⊗ → ⊙
  - Pod-to-Pod on same Node (W-7): encryption in namespace ⊘ → ⊙
- Security requirements: 10 of 12.
- SQuaRE characteristics: 8 of 10.

Conclusion

- Original goal: transparent encryption of Pod-to-Pod traffic in Project
- SecShift passed seven steps, from technology examination up to evaluation
- Result: reference implementation, evaluation and review prove design as a feasible and valuable security improvement ✓
- Future work: hardening of daemons, SMd-based distributed setup, hardware-based memory protection

Conclusion

- Original goal: transparent encryption of Pod-to-Pod traffic in Project
- SecShift passed seven steps, from technology examination up to evaluation
- Result: reference implementation, evaluation and review prove design as a feasible and valuable security improvement ✓
- Future work: hardening of daemons, SMd-based distributed setup, hardware-based memory protection

Conclusion

- Original goal: transparent encryption of Pod-to-Pod traffic in Project
- SecShift passed seven steps, from technology examination up to evaluation
- Result: reference implementation, evaluation and review prove design as a feasible and valuable security improvement ✓
- Future work: hardening of daemons, SMd-based distributed setup, hardware-based memory protection

Conclusion

- Original goal: transparent encryption of Pod-to-Pod traffic in Project
- SecShift passed seven steps, from technology examination up to evaluation
- Result: reference implementation, evaluation and review prove design as a feasible and valuable security improvement ✓
- Future work: hardening of daemons, SMd-based distributed setup, hardware-based memory protection

These slides and the associated thesis with further references will be published on my website `https://dpataky.eu` and are licensed as CC BY-SA 4.0

That's it. **Thanks!**

Questions? Feedback?

Improvements?

And don't forget: there is no Cloud – there's just somebody else's computer.

OpenShift networking with overlay

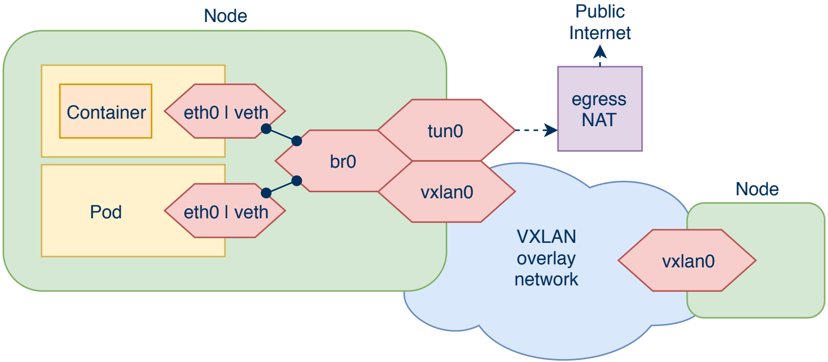

Figure 7: OpenShift networking

Security in OpenShift

- Kubernetes Namespaces (Projects)
- User management provides authentication (tokens) and authorisation (RBAC)
- API enforces TLS and offers Secrets storage
- Linux namespaces, SELinux, cgroups
- Security context constraints (SCCs) for Pods
- Extensions deliver more possibilities (Envoy proxies)

Traffic flows and adversaries

- Three traffic flows: on the same Node (TF1), through routers in the same data centre (TF2) and across DCs via the uplink (TF3)
- Four adversaries: passive attacker listening on routers (AM1), active attacker modifying routing configurations (AM2), misconfigurations in log collections (AM3) and attackers accessing Secret data (AM4)
- Identifying weaknesses in components and interconnections based on gathered attack surfaces

Domain model

Extended topology for *true* distribution

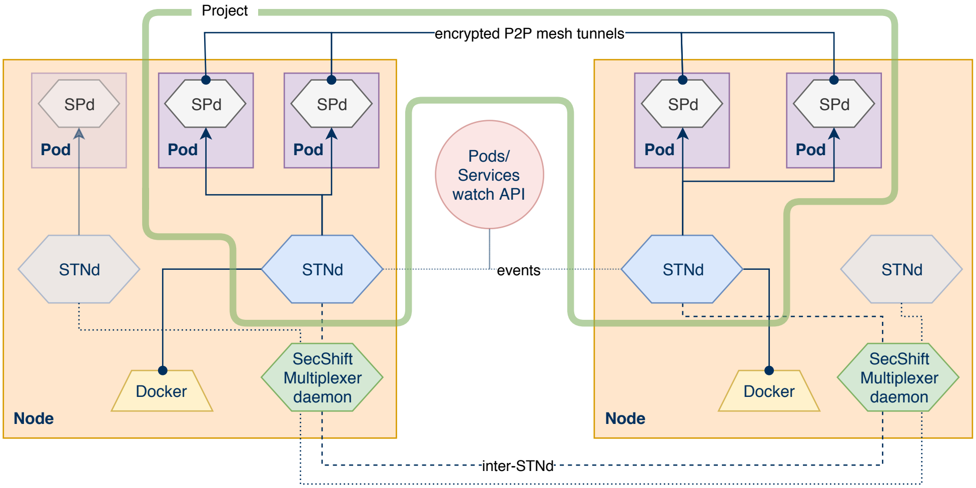

Figure 8: SecShift distributed design topology with SMd (and no usage of Secret or APIs)

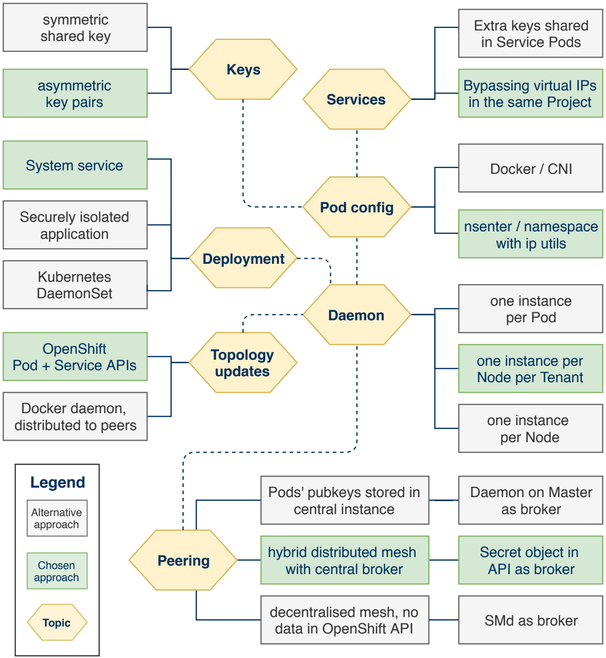

Figure 9: Design alternatives and choices

Daemon tasks

Daemon	Tasks
STNd	Mesh with STNd peers: channels for heartbeat and exchange
	Local Pods: list of local Pods and their public keys
	Remote Pods: with public keys, received from peers
	Topology: updates in the Project topology (from API, Pods/Services)
	Secret: updates from other STNd peers
	Docker: container details, events on the local Node
SPd	Coupling: channel to STNd, listening for commands
	Key: creation and updates, sending public key to STNd
	Network: network configuration (routes, NAT)
	DNS: proxy DNS to connect Pods directly instead of Service IPs
(SMd)	Multiplexer: route packets from STNds to remote peers
	Static connection: keeps one long-living channel to each Node

Table 2: List of tasks for all daemons, including the SMd

STNd peer announcement

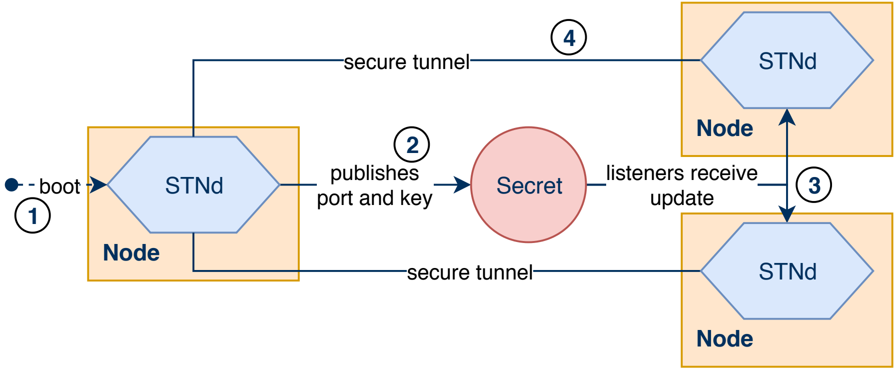

Figure 10: Daemons announce themselves to their peers

Reference implementation

- OpenShift test cluster with master and four nodes
- All nodes in WireGuard mesh network
- Daemons in Go, encryption interface with WireGuard
- Transparent routing: namespace-local network policies, routing tables, iptables rules
- Bypassing of Service IPs by proxying DNS and modifying answers

Demo

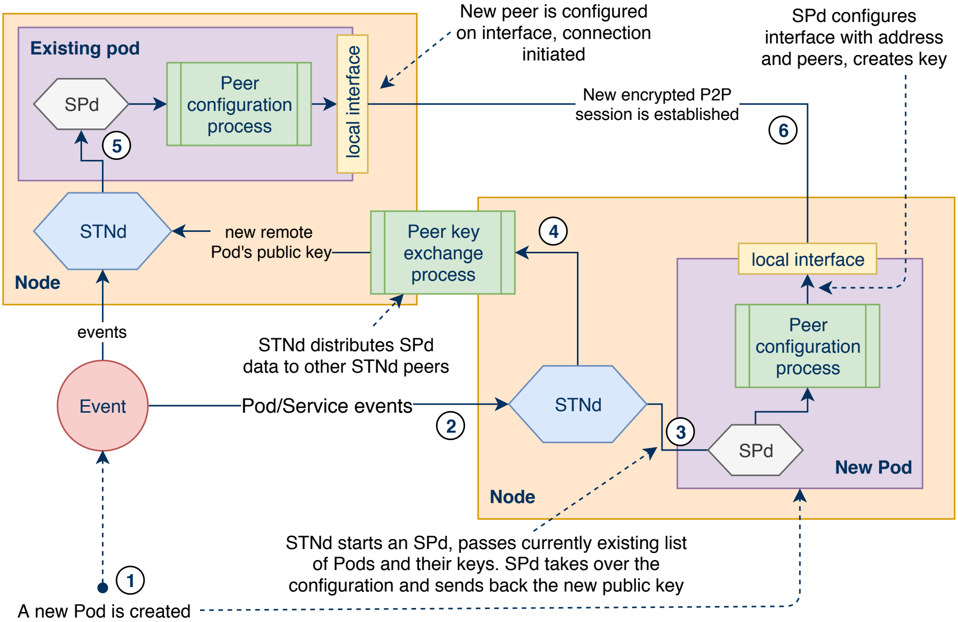

Figure 11: Daemons exchange keys in distributed setup

Throughput in secured setup with UDP

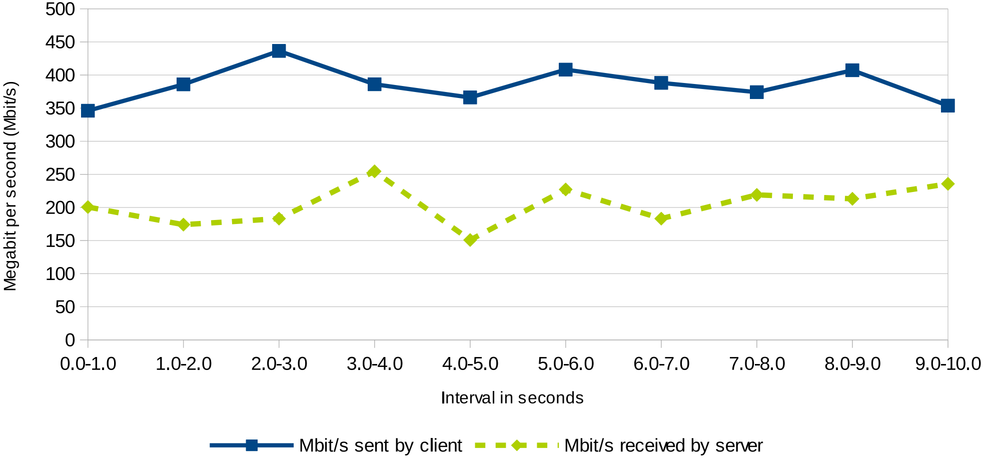

Figure 12: iperf with secured cross-Node tunnels, UDP, 1000Mbit/s bandwidth, no peer updates