Release: v1.0
Classification: RG-C0

Container Security Verification Standard

Version 1.0

Date: 2019-02-26

1 Table of Contents

1 TABLE OF CONTENTS 2
2 FRONTISPIECE 4
2.1 About the Standard 4
2.2 Copyright and License 4
2.3 Contribution 4
3 PREFACE 5
4 USING THE CSVS 6
4.1 Container Security Verification Levels 6
4.2 How to use this standard 6
4.2.1 Level 1: Opportunistic 7
4.2.2 Level 2: Standard 7
4.2.3 Level 3: Advanced 7
4.3 Applying CSVS in Practice 7
4.4 Use Cases 8
4.4.1 As Detailed Security Architecture Guidance 8
4.4.2 As a Replacement for Off-the-shelf Checklists 8
4.4.3 For Security Trainings 8
5 V1: ORGANIZATIONAL 9
5.1 Control Objective 9
5.2 Security Verification Requirements 9
6 V2: INFRASTRUCTURE 10
6.1 Control Objective 10
6.2 Security Verification Requirements 10
7 V3: CONTAINERS 12
7.1 Control Objective 12
7.2 Security Verification Requirements 12
8 V4: ORCHESTRATION MANAGEMENT 14
8.1 Control Objective 14
8.2 Security Verification Requirements 14
9 V5: IMAGE DISTRIBUTION 16
9.1 Control Objective 16
9.2 Security Verification Requirements 16
10 V6: SECRETS AND KEYS 17
10.1 Control Objective 17
10.2 Security Verification Requirements 17
11 V7: NETWORK 19
11.1 Control Objective 19
11.2 Security Verification Requirements 19
12 V8: STORAGE 21
12.1 Control Objective 21
12.2 Security Verification Requirements 21

13 V9: LOGGING & MONITORING 22
13.1 CONTROL OBJECTIVE 22
13.2 SECURITY VERIFICATION REQUIREMENTS 22
14 V10: INTEGRATION 23
14.1 CONTROL OBJECTIVE 23
14.2 SECURITY VERIFICATION REQUIREMENTS 23
15 V11: DISASTER RECOVERY 24
15.1 CONTROL OBJECTIVE 24
15.2 SECURITY VERIFICATION REQUIREMENTS 24
16 V12: TESTING 25
16.1 CONTROL OBJECTIVE 25
16.2 SECURITY VERIFICATION REQUIREMENTS 25
17 APPENDIX A: GLOSSARY 27
18 APPENDIX B: REFERENCES 28

2 Frontispiece

2.1 About the Standard

The Container Security Verification Standard is a list of security requirements or tests that can be used by architects, developers, testers, security professionals and even consumers to define what a secure container and corresponding infrastructure is.

2.2 Copyright and License

The code used to build the CSVS document and structure is based on the OWASP ASVS project.

2.3 Contribution

If you find any issues within the standard that should be addressed:

- Design of the standard.
- Missing controls.
- Ineffective or outdated controls.
- Unclear wording, spelling, grammar issues.
- Formatting issues.
- Translation issues – if a control's wording is such that trying express it in your language will be difficult or impossible, please let us know. If it doesn't work in Spanish or Thai, it probably isn't working in English either.
- Offers of translation – please let us know so that we can direct you to folks already working on your language.

Please log an issue here: https://github.com/redguard/CSVS/issues

Lead Authors	Contributors & Reviewers
Sven Vetsch	Alexander Hermann, Dominik Nufer, Dominique Meier, Patrick Schmid, Daniel Tschabold

3 Preface

Welcome to the Container Security Verification Standard (CSVS) version 1.0. The CSVS is a community-effort to establish a framework of security requirements and controls that focus on normalizing the functional and non-functional security controls required when designing, developing and testing container-based solutions with a focus on Docker.

We expect that there will most likely never be 100% agreement on this standard. Risk analysis is always subjective to some extent, which creates a challenge when attempting to generalize in a one size fits all standard. We're always happy to hear your feedback and re-evaluate the requirements.

4 Using the CSVS

CSVS has two main goals:

- Help organizations develop and maintain secure containers and container infrastructure.
- Allow security services, security tool vendors, and consumers to align their requirements and offerings.

4.1 Container Security Verification Levels

The Container Security Verification Standard defines three security verification levels, with each level increasing in depth.

- CSVS Level 1 is meant for all container projects.
- CSVS Level 2 is meant for container projects that deal with sensitive data or business logic, which requires additional protection.
- CSVS Level 3 is meant for the most critical container projects that perform high value transactions, contain sensitive personal or medical data, or any container that requires the highest level of trust.

Each CSVS level contains a list of security requirements. Each of these requirements can also be mapped to security-specific features and capabilities that must be built into the containers or their underlying infrastructure.

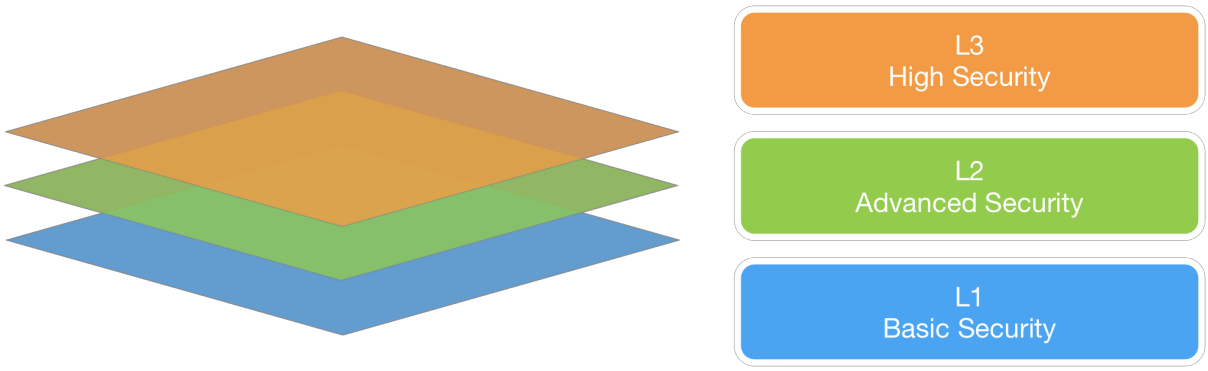

Figure 1 - Container Security Verification Standard levels

4.2 How to use this standard

One of the best ways to use the Container Security Verification Standard is to use it as blueprint to create a security checklist specific to your project, platform or organization. Tailoring the CSVS to your use cases will increase the focus on the security requirements that are most important to your projects and environments.

4.2.1 Level 1: Opportunistic

A container-based infrastructure/solution achieves CSVS Level 1 (or Opportunistic) if it adequately defends against well known security threats that are easy to discover and easy to abuse.

Level 1 is typically appropriate for applications where low confidence in the correct use of security controls including availability is required, or to provide a quick analysis of a fleet of enterprise applications, or assisting in developing a prioritized list of security requirements as part of a multi-phase effort. We consider Level 1 the minimum required for all container projects. If data processed by your container-based solution has high value, you would rarely want to stop at a Level 1 review.

4.2.2 Level 2: Standard

A container-based infrastructure/solution achieves CSVS Level 2 (or Standard) if it adequately defends against most of the risks associated with container-based solutions today.

Level 2 ensures that security controls are in place, effective/tested, and used within the whole solution. Level 2 is typically appropriate for container-based projects that handle significant and sensitive transactions, including those that process confident information, implement business-critical or sensitive functions, or process other sensitive assets.

4.2.3 Level 3: Advanced

CSVS Level 3 is the highest level of verification within the CSVS. This level is typically reserved for container-based solutions that require significant levels of security verification, such as those that may be found within areas of military, health and safety, critical infrastructure, etc.

Organizations may require CSVS Level 3 for applications that perform critical functions, where failure could significantly impact the organization's operations, and even its survivability. A container-based solution achieves CSVS Level 3 (or Advanced) if it adequately defends against advanced adversaries and also demonstrates principles of good security design.

An application at CSVS Level 3 requires more in depth analysis, architecture, coding, and testing than all the other levels. A secure container infrastructure is modularized in a meaningful way (to facilitate e.g. resilience, scalability, and most of all, layers of security), and each module (separated by network connection and/or physical instance) takes care of its own security responsibilities (defense in depth), that need to be properly documented and tested. Responsibilities include controls for ensuring confidentiality (e.g. encryption), integrity (e.g. transactions, input validation), availability (e.g. handling load gracefully), authentication (including between systems), non-repudiation, authorization, and auditing (logging).

4.3 Applying CSVS in Practice

Different threats have different motivations. Some industries have unique information and technology assets and domain specific regulatory compliance requirements. Although some unique criteria and some differences in threats exist for each industry, a common theme throughout all industry segments is that opportunistic attackers will look for any easily exploitable vulnerabilities, which is why CSVS Level 1 is recommended for all container-based projects regardless of industry.

Organizations are strongly encouraged to look more deeply at their unique risk characteristics based on the nature of their business. At the other end of the spectrum is CSVS Level 3, which is reserved for those cases that might endanger human safety or when a full application breach could severely impact the organization.

4.4 Use Cases

4.4.1 As Detailed Security Architecture Guidance

One of the more common uses for the Container Security Verification Standard is as a resource for security architects. The two major security architecture frameworks, SABSA and TOGAF, are missing a great deal of information that is necessary to complete application and container security architecture review. CSVS can be used to fill in those gaps by allowing security architects to choose better controls for common problems.

4.4.2 As a Replacement for Off-the-shelf Checklists

Many organizations can benefit from adopting the CSVS, by choosing one of the three levels, or by forking CSVS and changing what is required for each risk level in a domain specific way. We encourage this type of forking as long as traceability is maintained.

4.4.3 For Security Trainings

The CSVS can also be used to define characteristics of secure container infrastructure. Many security courses are simply ethical hacking courses with a light smear of operational tips. Security trainings can use the CSVS with its strong focus on the proactive controls to tech about best practices.

5 V1: Organizational

5.1 Control Objective

In a perfect world, security would be considered throughout all phases of development. In reality however, security is often only a consideration at a late stage in the SDLC. Besides the technical controls, the CSVS requires processes to be in place that ensure that the security has been explicitly addressed when planning the architecture of the solution, and that the functional and security roles of all components are known.

The category "V1" lists requirements pertaining to processes, architecture and design of the container solution. As such, this category can't be mapped to technical test cases but must be tackled on a process level. In addition, it should be noted, that this category is not and will never be complete as each organization is structured differently and each setup has different organizational and process requirements. The CSVS only tries to cover some basic aspects in this category and it's highly recommended to extend the requirements based on your needs.

5.2 Security Verification Requirements

Please note that the requirements in this section are non-exhaustive as many organizational security controls aren't solely focused on container infrastructures.

#	Description	L1	L2	L3	Since
1.1	Verify that technical employees (especially the ones tasked with DevOps like activities and architects) receive regular training on security aspects of the technologies they use.	✓	✓	✓	1.0
1.2	Verify that managers receive regular training on security aspects of the technologies used in their projects.			✓	1.0
1.3	Verify that all handled data is classified based on internal data classification standards.	✓	✓	✓	1.0
1.4	Verify that each service/application (can consist of multiple containers) has a security concept which provides information on the security needs of the service/application and how they are or will be addressed.		✓	✓	1.0
1.5	Verify that identified security risks and vulnerabilities are promptly eliminated (or an exception is granted) and centrally managed according to a predefined risk and vulnerability management process.		✓	✓	1.0
1.6	Verify the roles and responsibilities concerning the container infrastructure are defined. This includes e.g. who approves connectivity or decides on allowed base images.		✓	✓	1.0

6 V2: Infrastructure

6.1 Control Objective

The underlying infrastructure can be very different for various setups but it's the basis of each and must therefore provide the possibility for the upper layers to achieve the demanded level of security.

Ensure that a verified container solution satisfies the following high level requirements:

- Ensure that the infrastructure provides adequate resources.
- Harden the base infrastructure including the container platform.

6.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
2.1	Verify that the overall architecture and design including networking inside and outside of the container solution is defined.	✓	✓	✓	1.0
2.2	Verify that the infrastructure, including all components thereof (nodes, networks, containers, ...) are documented (ideally fully automated).	✓	✓	✓	1.0
2.3	Verify that all of the used components are supported/maintained and compatible with each other (OS, Docker Engine, UCP, DTR, ...).	✓	✓	✓	1.0
2.4	Verify that adequate resources are allocated to all nodes for them to run stable.	✓	✓	✓	1.0
2.5	Verify that the resources available to containers are limited (ulimit).		✓	✓	1.0
2.6	Verify that SELinux or AppArmor is enabled and running on all nodes as well as for *dockerd*.			✓	1.0
2.7	Verify that updates for both the nodes and the Docker Engine running on them are applied in regular intervals. Ideally, applying updates is fully automated.	✓	✓	✓	1.0
2.8	Verify that updates are rolled out using a canary deployment/release strategy, which allows rollbacks.		✓	✓	1.0
2.9	Verify that *dockerd* is configured with *live restore* enabled.		✓	✓	1.0
2.10	Verify that permissions to the configuration of *dockerd* is restricted to users that actually need access to it and are properly logged.	✓	✓	✓	1.0

2.11	Verify that all nodes undergo regular automated security scans which cover the whole operating system and not just container related elements.		✓	✓	1.0
2.12	Verify that container-specific operating systems (e.g. Container Linux, RancherOS, RedHat Project Atomic, VMware Photon) are used on all nodes instead of general-purpose ones.			✓	1.0
2.13	Verify that all nodes are hardened based on common best practices.	✓	✓	✓	1.0
2.14	Verify that unless otherwise specified, the default Docker configuration values are used.	✓	✓	✓	1.0
2.15	Verify that direct access to nodes (e.g. via SSH or RDP) is restricted as much as possible.	✓	✓	✓	1.0

7 V3: Containers

7.1 Control Objective

The main component of container-based solutions are the containers themselves. Not only do they contain services and application logic but also interact with other systems and containers to exchange data that is often sensitive and demands accurate protection.

Ensure that a verified container solution satisfies the following high level requirements:

- Ensure that the containers run with the least possible privileges.
- Harden services inside the container and minimize the attack surface.
- Leverage security features of the container technology in use.

7.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
3.1	Verify that the root user isn't used within containers except during initialization and privileges are dropped on completion.		✓	✓	1.0
3.2	Verify that user namespacing is enabled.		✓	✓	1.0
3.3	Verify that within each container image, a new user is created, which is then used to perform all operations within the container.			✓	1.0
3.4	Verify that a specific (non-standard) *seccomp*-profile is applied to each container-based on the needs of the container.			✓	1.0
3.5	Verify that containers cannot be granted any additional privileges during their runtime (`--no-new-privileges` flag).	✓	✓	✓	1.0
3.6	Verify that all base images are explicitly specified, using their hash instead of name and tag.			✓	1.0
3.7	Verify that the signature of each image is verified before productive usage.			✓	1.0
3.8	Verify that only required software packages are installed in images.	✓	✓	✓	1.0
3.9	Verify that the root file system is mounted in read-only mode.		✓	✓	1.0

3.10	Verify that after a container has been actively accessed (e.g., for troubleshooting), it's deleted and replaced by a new instance (container) of the image.		✓	✓	1.0
3.11	Verify that Dockerfiles use the `COPY` directive instead of the `ADD` directive unless the source is fully trusted.	✓	✓	✓	1.0
3.12	Verify that remote management services such as SSH or RDP are disabled or not even installed within containers.	✓	✓	✓	1.0
3.13	Verify that exposed services such as *etcd* are either only available to fully trusted systems or require authentication.	✓	✓	✓	1.0
3.14	Verify that the number of allowed processes within a container is precisely defined and limited to this value by using `--pids-limit`.			✓	1.0
3.15	Verify that the Docker socket isn't mounted inside any container unless they are used for monitoring or administration. If access to the Docker socket is required, check if read-only access is sufficient and limit the access of the container accordingly.	✓	✓	✓	1.0

8 V4: Orchestration Management

8.1 Control Objective

As your container-based solutions outgrows a certain amount of nodes and containers an orchestrator is needed. The orchestrator helps managing and administrating the solution and keep track of what's going on. As the orchestrator is such a mighty central piece in a container infrastructure the security level of the orchestrator directly affects every other aspect of your infrastructure.

Ensure that a verified container solution satisfies the following high level requirements:

- Uptime for the orchestrator is guaranteed.
- The orchestrator is hardened.
- Interaction with the orchestrator is mostly automated to avoid human errors.

8.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
4.1	Verify that manager nodes are set up redundant and ready to support high availability.	✓	✓	✓	1.0
4.2	Verify that an odd number of manager nodes is deployed with a minimum of three nodes.	✓	✓	✓	1.0
4.3	Verify that managers are distributed across multiple data centers and availability zones.		✓	✓	1.0
4.4	Verify that manager nodes run with *auto-lock* enabled.			✓	1.0
4.5	Verify that the orchestrator rebalances the active containers on a regular basis.	✓	✓	✓	1.0
4.6	Verify that manager nodes don't take on worker tasks and containers.	✓	✓	✓	1.0
4.7	Verify that predefined labels are used to properly identify and manage all resources.		✓	✓	1.0
4.8	Verify that containers that are no longer needed are deleted.		✓	✓	1.0
4.9	Verify that only containers with the same data classification level run on the same node.			✓	1.0

4.10	Verify that only containers with the same level of exposure (e.g. Internet facing) run on the same node.	✓	1.0

9 V5: Image Distribution

9.1 Control Objective

To have some running containers first you need images to be built. Images define what will be running inside of a container and for example which version of a software package will be used. As the base of potentially a huge amount of container, the security of each image is essential to ensure safe operation of an environment.

Ensure that a verified container solution satisfies the following high level requirements:

- Images are hardened.
- No sensitive data is stored inside of images.
- Images are checked for vulnerable components.

9.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
5.1	Verify that an odd number of image registries (e.g., DTR) with a minimum of three registries is used.			✓	1.0
5.2	Verify that garbage collection is enabled on the image registries and running on a regular basis.	✓	✓	✓	1.0
5.3	Verify that all images undergo regular automated security scans.		✓	✓	1.0
5.4	Verify that containers are always created based on the most recent corresponding image and not local caches.	✓	✓	✓	1.0
5.5	Verify that all images are using tags whereas only production/master is allowed to use the default *latest* tag.		✓	✓	1.0

10 V6: Secrets and Keys

10.1 Control Objective

Production systems are usually using some kind of secrets and cryptographic keys. Those can be used for configuration purposes and include usernames and passwords or to allow the protection of information by cryptographic means. This section defines how such sensitive information can should be handled.

Ensure that a verified container solution satisfies the following high level requirements:

- Protect sensitive information.
- Verify secure handling of cryptographic material.
- Rotate cryptographic keys on a regular basis.

10.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
6.1	Verify that an RBAC model to manage access control is used.		✓	✓	1.0
6.2	Verify that Docker Content Trust is enabled and enforced.			✓	1.0
6.3	Sensitive information may never be part of a Dockerfile or Docker-Compose file. In particular, verify that e.g. Docker secrets are used for handling sensitive information like API keys and passwords.	✓	✓	✓	1.0
6.4	Verify that orchestration join keys are rotated in regular intervals.		✓	✓	1.0
6.5	Verify that auto-lock keys are rotated in regular intervals if auto-lock is enabled.		✓	✓	1.0
6.6	Verify that node certificates are rotated in regular intervals.		✓	✓	1.0
6.7	Verify that CA certificates are rotated in regular intervals.		✓	✓	1.0
6.8	Verify that your own CA is used for generating and verifying certificates used for mutual TLS authentication of inter-cluster communication.		✓	✓	1.0
6.9	Verify that the SSL/TLS certificates used (e.g. for UCP and DTR) are validated.	✓	✓	✓	1.0

6.10	Verify that secrets (e.g. cryptographic keys and passwords) are used securely with a secret management solution instead of e.g. exposed to a container by using environment variables.	✓	✓	1.0

11 V7: Network

11.1 Control Objective

Nearly all modern applications and services aren't monolithic but instead consist of multiple components interacting with each other through network connections. Securing networks is its own security discipline but there are some aspects that container technologies can affect and where they can improve security when using networks.

Ensure that a verified container solution satisfies the following high level requirements:

- Choose a good network driver and configure it correctly.
- Disable unneeded features and apply restrictions.
- Enforce encryption when transferring data over networks.

11.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
7.1	Verify that a production ready networking driver is used.	✓	✓	✓	1.0
7.2	Verify that load balancing features are activated (e.g. by using DNS Round Robin or virtual IPs (VIP)).		✓	✓	1.0
7.3	Verify that the Docker userland proxy (which is enabled by default) is disabled.		✓	✓	1.0
7.4	Verify that the default bridge (*docker0*) is not used.	✓	✓	✓	1.0
7.5	Verify that *dockerd* is configured in a way that network communication between different containers is not possible by default. This can be done either by not using the *docker0* bridge or setting *--icc* to false.		✓	✓	1.0
7.6	Verify that *dockerd* is permitted to modify *iptables* rules.	✓	✓	✓	1.0
7.7	Verify that published ports are assigned to a specific network interface of a node.	✓	✓	✓	1.0
7.8	Verify that management and data/application traffic is separated by different network interfaces.			✓	1.0

7.9	Verify that each application (one or more services) is assigned at least one separate, isolated overlay network in order to ensure Layer 3 segmentation.		✓	✓	1.0
7.10	Verify that encryption between containers or nodes on the overlay network is enabled.		✓	✓	1.0
7.11	Verify that the used subnets do not overlap with other subnets (e.g. overlay networks).	✓	✓	✓	1.0
7.12	Verify that published ports are limited to a necessary minimum.	✓	✓	✓	1.0

12 V8: Storage

12.1 Control Objective

As containers are ephemeral it's important to provide a reliable and secure storage backend for persistent data. Not only is the availability of stored data essential but also its integrity and access control measures.

Ensure that a verified container solution satisfies the following high level requirements:

- Choose a good storage driver and configure it correctly.
- Make sure data is not locally stored on nodes for persistence.

12.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
8.1	Verify that a production ready storage backend is used.	✓	✓	✓	1.0
8.2	Verify that the image storage backend is redundant and located in a secured network zone.	✓	✓	✓	1.0
8.3	Verify that a suitable and tested data storage driver is used in order to ensure the replication and availability of application data.	✓	✓	✓	1.0
8.4	Verify that persistent data is never stored directly inside a container, but on a corresponding docker volume or mount point instead.	✓	✓	✓	1.0
8.5	Verify that persistent data is regularly backed up according to a suitable well defined backup concept and the restore is tested.	✓	✓	✓	1.0

13 V9: Logging & Monitoring

13.1 Control Objective

Securing containers and infrastructure is one thing but making sure that you know when things go wrong is no less important. Logging and monitoring provide you insights in the current state of your solution and allow you to react accordingly.

Ensure that a verified container solution satisfies the following high level requirements:

- Have a central logging and monitoring instance.
- Monitor all your components.

13.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
9.1	Verify that the underlying system, Docker Engine, as well as containers and their processes are logged.		✓	✓	1.0
9.2	Verify that the used resources at both node and container level are monitored.		✓	✓	1.0
9.3	Verify that the storage backend is monitored.		✓	✓	1.0
9.4	Verify that Docker's health checking functionality is used for all containers and their status is monitored.		✓	✓	1.0
9.5	Verify that all logs are transferred and stored to/in a central location.		✓	✓	1.0
9.6	Verify that in production environments, the log level of *dockerd* is set to *info*.	✓	✓	✓	1.0

14 V10: Integration

14.1 Control Objective

Container-based solutions are normally not self-contained but instead integrate with a variety of different systems. Such systems could be IAM solutions, CI/CD pipelines or existing network environments. Any interaction also poses a potential threat to a container-based solution and vice-versa.

Ensure that a verified container solution satisfies the following high level requirements:

- Integrate into existing security infrastructure.
- Place information in a central inventory and change management system.
- Leverage existing networking infrastructure.

14.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
10.1	Verify that the orchestration solution (e.g. UCP) and registry (e.g. DTR) are integrated into the existing infrastructure (SSO, DCT,...).		✓	✓	1.0
10.2	Verify that the CI/CD tools and systems are connected to the Docker infrastructure to enable changes in nodes, images, or the network to be tested and rolled out fully automated.		✓	✓	1.0
10.3	Verify that additional nodes can be set up automatically (e.g., Puppet, Chef, Ansible, Salt, Terraform) and configured the same way as existing nodes.		✓	✓	1.0
10.4	Verify that a central change management system is implemented and all changes to the container infrastructure and its components are tracked there.		✓	✓	1.0
10.5	Verify that a discovery and registration service like consul, Zookeeper, Eureka, Etcd or even just DNS is used internally and externally.		✓	✓	1.0
10.6	Verify that users and roles are mapped to an existing central IAM solution.			✓	1.0

15 V11: Disaster Recovery

15.1 Control Objective

When things go wrong it is important to get back on your feed as fast as possible without compromizing on security. This category describes requirements on how to verify that your disaster recovery works as expected and downtime is kept short.

Ensure that a verified container solution satisfies the following high level requirements:

- Backups are created on a regular basis.
- Restoring steps are automated.
- Self-healing capabilities are leveraged.

15.2 Security Verification Requirements

#	Description	L1	L2	L3	Since
11.1	Verify that regular backups (UCP, DTR, and Swarm) are performed. A weekly backup has to be performed at a minimum.	✓	✓	✓	1.0
11.2	Verify that the restoration of the infrastructure is automated, documented and regularly tested.	✓	✓	✓	1.0
11.3	Verify that upgrades and downgrades of the basic infrastructure as well as the Docker Engine is automated, documented and regularly tested.			✓	1.0
11.4	Verify that the recovery of individual applications/services is automated, documented and regularly tested.		✓	✓	1.0
11.5	Verify that an `on-failure` restart policy is enabled for each container.		✓	✓	1.0

16 V12: Testing

16.1 Control Objective

Technology is always moving forward and steadily changing in unexpected ways. Based on this securing a container-based solution isn't just a one-time effort, but different checks and validations should be done on a regular basis.

Ensure that a verified container solution satisfies the following high level requirements on a regular basis:

- Recovery from failure.
- Ensure that security settings are taking effect.
- Documentation of the current state of the container-based solution.

16.2 Security Verification Requirements

The following requirements are to be evaluated based on a regular execution cycle. It is recommended to keep the cycle at three months or shorter.

#	Description	L1	L2	L3	Since
12.1	Verify that all user and group permissions to resources are in line with the specifications and documentation.	✓	✓	✓	1.0
12.2	Verify that application/container resource limitations work as defined.		✓	✓	1.0
12.3	Verify that each service can be successfully recreated in a fully automated way.		✓	✓	1.0
12.4	Verify that certificates and keys are rotated according to the specifications.			✓	1.0
12.5	Verify that the configurations, images and networks of all services can be updated and downgraded on a rolling basis.		✓	✓	1.0
12.6	Verify that nodes as well as the Docker Engine are up to date.	✓	✓	✓	1.0
12.7	Verify that the load-balancing strategies work as defined.		✓	✓	1.0
12.8	Verify that containers are balanced across the cluster based on the defined strategy.		✓	✓	1.0

12.9	Verify that all services can recover from failures of nodes and individual containers.		✓	✓	1.0
12.10	Verify that backups can be restored for all services in the event of a total failure.	✓	✓	✓	1.0
12.11	Verify that *Docker Security Bench* runs regularly and passes.		✓	✓	1.0

17 Appendix A: Glossary

- **API** – Application programming interface
- **CA** – Certificate Authority
- **CI/CD** – Continuous Integration and Continuous Delivery
- **CLI** – Command Line Interface
- **DCT** – Docker Content Trust
- **DNS RR** – DNS Round Robin
- **DTR** – Docker Trusted Registry
- **IAM** - Identity and Access Management
- **LDAP** – Lightweight Directory Access Protocol
- **mTLS** – Mutual TLS (authentication)
- **Node** – A physical host on which the container / Docker engine runs. Could be a single host or a host as part of a cluster.
- **RBAC** – Role-Based Access Control
- **RDP** – Remote Desktop Protocol
- **Service** – A service which runs in a container.
- **SSH** – Secure Shell
- **SSL** – Secure Sockets Layer
- **SSO** – Single Sign-On
- **TLS** – Transport Layer Security, the successor to Secure Sockets Layer (SSL)
- **UCP** – Docker Universal Control Plane
- **VIP** – Virtual IP

18 Appendix B: References

The following projects are most likely to be useful to users/adopters of this standard and were also helpful during its creation:

- OWASP Application Security Verification Standard - https://www.owasp.org/index.php/ASVS
- CIS Docker Benchmark - https://www.cisecurity.org/benchmark/docker/
- Docker Reference Architectures - https://success.docker.com/architectures
- Docker EE Best Practices and Design Considerations - https://success.docker.com/article/docker-ee-best-practices
- NIST Application Container Security Guide - https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-190.pdf