# Methodology

Auszug aus der Roh-Form meiner Bachelorarbeit, wird noch ausformuliert & um unprofessionelle Formulierungen bereinigt ;)

For a concise view of the methodology, consult the table “Risk assessment formula”

difficulties with:

A how generically should vectors be set?

B how to structure vectors (into categories?)

Solution to A: vectors split and merged after risk assessment sketch; if there were considerable differences in the estimated values, they were split. If no diffs, they were merged.

Solution to B: Considerable time invested, no optimal solution was found. Ultimately ignored this, since more time investment wasnt feasible or added much value to the goals. TODO: Explain process, what was looked at and why it wasnt good, how we arrived at the end structure.

Risk assessment formula:

Risk = Probability \* Impact.

Impact was taken as single value of 1 through 3 and estimated through None/Theoretical (0), Low/Intermediate-Step (1), non-severe security principle violation (2), severe security principle violation (3)

Probability was a bitch to define properly. Therefore split into four factors:

Vantage Point, Required Access Level (RAL), Detectability and Exploitability.

Those initially had values between 0 though 3, but outlier values of 4 were defined for RAL and Vantage Point (in sync with existing assessment methodologies within HvS).

The average of these four values is taken as the total propability value, ranging from 0.25 through 3.5 (low <= 1.25, medium <= 2.25).

Vantage Point:

physical access (0, see above since your own or the cloud providers hardware-accessing employees can also do whatever they want); node or management-interface (1); within container (2); within company network (3); from public www (4)

Required Access Level:

cloud/infrastructure admin (0, since a rogue employee with super-admin can do whatever they want and this is about baseline security); cluster/system-admin (1); cluster/system user with read/write access (2); cluster/system user with read-only access (3); unauthenticated (4)

Detectability:

Difficult (1) since it needs custom tools for environment-specific vuln detction; Average (2) since it is either generic but needs simple custom tools or its individualized but can be identified with some slight tool individualization; Easy (3) since there are generic script-kiddie tools / GUI-paths to find the vuln

Exploitability:

Theoretical (0, since this is the level of unpublished 0-days and we are still doing baseline security); difficult (1) needs custom tools for environment-specific exploitation; Average(2), since its a generic exploit but needs simple custom tools or its inividualized but can be exploited with some slight tool individualization; Easy (3) since there are generic script-kiddie tools / GUI-paths to exploit the vuln

This leads to a total risk of 0 through 10.5, which is then rounded to full integers and capped at 10. In accordance to HvS internal models, total risk values <= 3 are defined as low, <= 6 as medium and values above that are defined as high.

specific values for each vector are estimated in a context with multiple assumptions:

TODO: callback to assumptions in scope limitation

- If multiple techniques can be used / impacts can occur to leverage a vector, all factor values of the one with the highest total risk are taken

- Values might decrease through the implementation of security measures, leading to a lower total value. (If multiple techniques could be used to leverage a vector and only one gets its total risk reduced, the new maximum risk value of that vector becomes the vector value(s)

goal is to reduce values above threshold X to below threshold X by applying security measures. This aims to ensure a basic security level, not something against APT groups / zero-day protection / targeted attacks with a lot of resources and competence. (no online banking, user data of average confidentiality etc)

# Vectors

## Vector definition

|  |  |  |
| --- | --- | --- |
| V-ID | Vector | Notes |
| V01 | Reconaissance through Kubernetes & platform control plane interfaces | Gather information useful for further attacks through accessible: the Kubernetes dashboard & apiserver as well as potential platform webinterfaces & apiserver(s) |
| V02 | Read confidentials through Kubernetes & platform control plane interfaces | Gather confidential information through the Kubernetes dashboard & apiserver as well as potential platform webinterfaces & apiserver(s) |
| V03 | Change configuration through Kubernetes & platform control plane interfaces | Change the existing configuration through the Kubernetes dashboard & apiserver as well as potential platform webinterfaces & apiserver(s) |
| V04 | Compromise internal k8s control plane components (etcd, scheduler, controller-manager) | This vector comprises reconaissance, leaks of confidentials and configuration changes through Kubernetes components not intended to be accessible: etcd stores, kube-scheduler and kube-controller-manager |
| V05 | Supply compromised container (base) image | Supplying a malicious container image leading to security violations on the cluster (remote access for an attacker, resource misuse, data leakage, …). Most easily done untargeted (dockerhub images or dockerfiles on tutorials/help forums), but can be done targeted, too. Additionally, an image build process typically runs as root, leading to compromise possibilities to compromise the node where an image is built from a rogue dockerfile. (TODO: provoking stale image usage to exploit vulns, too?) |
| V06 | Supply compromised k8s configuration | Supplying a malicious kubernetes configuration leading to security violations on the cluster (remote access for an attacker, resource misuse, data leakage, …). Most easily done untargeted (tutorials/ help forums), but can be done targeted, too. |
| V07 | Compromise other application components (lateral movement) | Once an attacker gains access to a container, he may try to access more lucrative application components or information, i.e. sniffing traffic or accessing databases or containers with more confidential information/traffic. |
| V08 | Container breakout (R/W, Privilege Escalation) | Once inside a container, an attacker may try to gain access to the underlying host by a multitude of means. This includes invoking syscalls, accessing the host file system and elevation priviledges within or outside of the container environment |
| V09 | Compromise local image cache | If the cached image of a container can be manipulated, another container (which might even seem to fulfill the same function) violating security principles could be started. |
| V10 | Modify running container | Once inside a container, an attacker may try to modify the container to exfiltrate data or better suit their needs for further intrusion |
| V11 | Hoard resources (sabotage) | The provided resources may be misconfigured or misused to disrupt service availability (i.e. through fork bombing or misconfiguration) |
| V12 | Misuse resources (cryptojacking) | The provided resources may be misconfigured or misused for financial gains (mining cryptocurrencies) |
| V13 | Add malicious container | A malicious container may be started within the cluster |
| V14 | Add malicious node | A malicious node may be added to the cluster |
| V15 | Bad user practice (outside of cluster) | This vector comprises user practices outside of the cluster that lead to risks within it. Examples include phishing, openly publishing keys/tokens to public code repositories and more. |
| V16 | Incufficient base infrastructure hardening | The underlying nodes could allow an attacker easy entry, even if the containers themselves are hardened. This includes Side-Channel attacks like Spectre & Meltdown |
| V17 | Entry through known, unpatched vulnerabilities | Every system has to be kept up to date with security patches. Publicly known vulnerabilities might otherwise be exploited, leading to potentially devastating violations of security principles |

## Risk estimation of vectors

### Openshift Container Platform (OCP)

For all values in one place, consult Table “Risk Assessment v4”

V01:

A user with access to the apiserver / webinterface(s) and read access can scout out information.

By default, each account (project admin or project user, but not cluster admin) can only see information about his own project, a cluster admin can see all namespaces.

This could show outdated software versions, running systems / containers / pods / user account privileges / misconfigurations and may support in planning and confirming effectiveness of further attacks.

The information gathering processes and interfaces are known and documented pretty well, but the information gathered has to be analyzed specific to the environment.

V02:

In addition to V01, a user with access to the apiserver / webinterface(s) and read access can gather confidential secrets like certs, tokens or passwords which are intended to be used by automated systems and/or users to authenticate themselves to cluster components and gain privileged access like pull/push images, trigger actions in other applications / containers, …

These can be gathered and used for further access by an attacker.

Kube-hunter is a readily available tool and checks for this automatically.

V03:

In adition to both V01 and V02, a user with access to the apiserver / webinterface(s) and write access can change configurations on the cluster. By default, each account (project admin or project user, but not cluster admin) can only change the configuration of namespaces resources (i.e. access to project-specific resources like pods, services, routes, but not cluster-global resources like nodes, SCCs or interface/authorization configurations).

The capabilities can be looked up through the API, what you can achieve with it has to be analyzed environment-specifically though.

V04:

Misconfiguration of internal Kubernetes components (accessible by systems it is not assigned to be accessible by) could lead to a full cluster compromise. The cluster configuration and all secrets / authorization credentials are stored in the etcd instance(s). One would need to seriously fuck up the setup, since OCP configures everything through ansible and you would have to knowingly change some internal settings not intended to be changed in order to achieve this. Configurations are maintained by red hat, meaning config changes will be applied in updates and additionally sent out to notify relevant people subscribed to those alerts.

Kube-hunter checks for misconfiguration, but can’t find any (non-false-positive) openings with default settings. Would need zero-day / known vuln in Microsoft or red hat configs

V05:

This has two facettes: it can be untargeted (image spraying) and targeted (compromising a specific image known to be used by the target).

The untargeted version needs the least access, since it simply needs a (free) dockerhub account to upload malicious images that could or couldn’t fulfil the function they are advertised to do. This is done in the hopes of someone downloading that image for use in his own environment, thus starting attacker-supplied containers within their cluster.

This could allow an attacker remote access to a container in the cluster and/or exfiltrate information.

Even without injecting malware, an attacker could mislabel old software versions as newer ones so software with known vulnerabilities is deployed because it is though to be up to date.

The targeted version could be specialized uploads to docker hub (similar to broad phishing vs. spear phishing) or “poisoning” an internal container image repository.

Image builds run as root, which could further be exploited – but this would need a vulnerability in the OCP / Azure build process.

These methods are publically known and both the docker container runtime and docker hub actively try to mitigate this, but malicious images are only deleted when reported by enough users and the security settings within the container runtime are not set by default.

Base containers and malware / known vulnerable versions are readily available from public sources, but need some technical expertise to plug together.

V06:

Similar to V05, this can be done either untargeted by spraying to tutorials / help forums or targeted, similar to spear phising.

If a cluster administrator does not fully analyze or understand the configuration he gets from public sources, the cluster could be compromised fully, i.e. by implementing backdoors through malicious containers with special access and ability to be remotely accessed by the attacker.

Examples are readily available from public sources, but need some technical expertise to plug together.

V07:

Once an attacker sits within a container, he can scan the network for other containers, hosts, services, apis or similar interfaces to further his access. By default, all containers in all projects (except master & infra components) are put in the same subnet, allowing everyone to communicate with anyone else.

This is especially troubling for securing an environment with multiple tenants – even if the DB is not publically accessible, unauthorized access can be leveraged by anyone in the cluster.

Scanning tools like nmap etc. to find components to talk to are readily available, but their results are cluster-specific (everyone runs something different). Therefore some technical expertise is needed to leverage the network access needed.

V08:

A deployed container poses the risk of allowing access to the node it is running on, thus allowing an attacker to “break out” of the container and perform actions on the node.

This poses a considerable threat, since any container may run on any node by default, allowing an attacker full access to any containers running on the node he controls, which will – especially over time – have a great chance to include containers belonging to other projects.

The OCP default settings limit the possibility of this dramatically, the risk lies more in organizations relaxing the defaults in favour of easy usability. (A majority of container images straight from docker hub require UID 0, which is denied by the default SCC ‘restricted’ in OCP during admission. This results in crashlooping and non-functional containers, developers would need to customize any image themselves. The easiest way to stop those problems this is to permit the default service account within a project access to the ‘privileged’ SCC permissions. This would significantly increase the risk of a container breakout!)

This is probably the most-talked about attack vector regarding containers, but techniques are not obviously documented and breakout methods would have to be customized to the restrictions applied within a cluster.

V09:

If you can swap out the cached container image on a host, the swapped-in version will run the next time this node spins up this container.

This is a very sneaky way to inject a malicious container, but within the default settings, access to the host file system is required.

Not well known and not entirely trivial to do (sneakily).

V10:

Instead of deploying a container with malicious contents, an attacker can try to modify and use an already running container to its needs by loading additional tools/binaries, changing configurations or exfiltrating data. This could be done through an RCE vuln, ssh access or others, just like any compromised linux machine.

-> Common sense to do this, same technical level as any command line interaction with a linux system.

V11:

With enough access or restrictions too lax, an attacker may be able to seriously halt the availability of all workloads processed by the cluster by misconfiguration, conducting DOS attacks or wiping nodes or cluster configurations. Since it is a complex system, finding the sabotaged component can take considerable know-how and time if done well, increasing the impact – especially in on-premise environments, where resources are limited.

Wiping is common sense, sabotaging the cluster in a complex and effective way may take deeper knowledge and be customized to the environment.

V12:

In contrast to V11, an attacker will try to be sneaky if done well.

The goal here is to (ab)use the computing resources not belonging to and payed for by him to achieve monetary gain though mining cryptocurrencies.

Cryptojacking is regularly cited as an up-and-coming attack, but to do it with a low risk of being detected needs some technical skill.

V13:

Instead of manipulating running containers, an attacker with user access and permissions to spin up containers may start their own ones. (BYOC – bring-your-own-container?)

This is still restricted by container admission restrictions on the user/project, but at least he can install all needed binaries beforehand and his shell doesn’t die whenever the underlying container might be stopped.

Doing this is common sense, as before some technical skill is required to prepare a malicious container

V14:

An attacker could try to add a malicious node to the cluster and inspect or manipulate data in or exfiltrate data from containers scheduled on it. Since any container may run anywhere, there is a high chance of all containers eventually being run on a given node over time, exposing the whole cluster to an attacker. This could be sped up by manipulating the reports of remaining resources on the node towards the scheduler.

By design, cluster administrator access is needed to add a node within OCP.

This technique is not talked about that much, but still available in public resources and possible in all clusters. Docs are publically available to add nodes to a cluster, basic linux server administration skills are needed to follow them.

V15:

This vector comprises user practices outside of the cluster that lead to risks within it. Examples include phishing, openly publishing keys/tokens to public code repositories, password reuse, scouting specific software or container images used, publishing logs with information valuable to an attacker and more.

Could be done targeted (i.e. specific OSINT) or untargeted through github crawlers, scanning account/password dumps, …

Whatever you get could be used to access the cluster with the permissions granted by service-/user-accounts or as a reconnaissance base for further attacks.

There are tools available to do this, using them effectively requires some technical skill.

V16:

The underlying nodes could allow an attacker easy entry, even if the containers themselves are hardened. This includes Side-Channel attacks like Spectre & Meltdown, open ports on the servers exposed by other stuff running on it, being available from the public www, …

Vector exists mostly to sink all “classic” infra security measures in it, since those are researched and available everywhere and very much not the focus of the thesis.

Among worst case: unauthenticated access to run commands which is hosted publically on the internet for anyone to access and indexed by shodan. Bye bye cluster.

(Too many scenarios to hypothesize here, I’ll just point the finger at conventional server & infra hardening standards and guidelines)

-> Well known, still needs some technical skill to find vulns and exploit them

V17:

Sinkhole vector for patch management. Would be a measure against every preceding vector otherwise.

Worst case could be anything, thus maximum risk. (See kubernetes CVE with 9.8 / 10)

To check for this is common sense, some technical skill may be needed to find and exploit unpatched stuff.

### Azure Kubernetes Service (AKS)

For all values in one place, consult Table “Risk Assessment v4”

V01:

Same as OCP, except accessible from anywhere (cloud, duh).

-> doesn’t change total risk value

V02:

Same as OCP, except accessible from anywhere (cloud, duh).

-> increases total risk value slightly, pushing it just over the edge from medium to high

V03:

Same as OCP, except accessible from anywhere (cloud, duh).

-> increases total risk value slightly, both are still rated high in the end

V04:

Same as OCP, except accessible from anywhere (cloud, duh).

The master components are updated, configured and maintained by Microsoft, only when a restart is required the cluster administrator has to trigger it manually.

-> increases total risk value slightly, both are still rated medium in the end

V05:

Exactly the same as OCP

V06:

Exactly the same as OCP

V07:

Same as OCP.

The worst AKS-specific problem with this is the mitigation. This risk is not clearly documented in the setup section of the documentation. If one stumbles upon this information in further sections of the docs after setting up his cluster, he might postpone or deny changing the setting to isolate different projects by default. This is because a full cluster rebuild is needed to change this setting!

V08:

Difference to OCP: containers can be run with UID 0 and more relaxed settings in general by default. User-namespace remapping not in place by default, vastly increasing the risk of a container breakout!

This is more on the usability>security side of things.

-> raises risk, jumping from medium to high.

V09:

Same as OCP.

V10:

Same as OCP.

V11:

Difference to OCP: you can easily spin up more resources in the cloud -> less impact

-> risk decreases by a considerable margin, high to medium

V12:

Difference to OCP: an attacker can easily spin up more resources in the cloud -> *more* impact

-> risk *increases* by a considerable margin, medium to high

V13:

Same as OCP, except accessible from anywhere (cloud, duh).

-> increases total risk value slightly, both are still rated medium in the end

V14:

Accessible from anywhere (cloud).

-> total risk value unchanged

In contrast to OCP, you can spin up additional nodes more easily in AKS if configured on creation, but to access/control/manipulate them you still need cluster administrator access.

A tutorial on getting ssh access is available, but that’s lengthy and not trivial.

V15:

Same as OCP, except accessible from anywhere (cloud, duh).

-> doesn’t change total risk value

V16:

Suprisingly, same as OCP (despite the azure promise of PaaS-we-manage-your-infra)!

That’s the case since security updates on nodes that require a reboot are not done automatically, but have to be triggered manually or configured to trigger automatically.

Remediation is far less work though.

V17:

Same as OCP. Even infra still needs user interaction to be patched, see preceding vector.