Securing Developer Workstations with Docker

Introduction

Cyber attackers are finding new ways to breach companies everyday. The number of security breaches and the cost to mitigate the fallout is escalating quickly. In 2021 the average cost of a US breach grew to more than $9.4M.

It's not just production environments that are under increased attack, it's test and increasingly development environments (e.g., a software developer's laptop). The reason is simple: humans are often the weakest link in a company's security chain.

In 2021 alone, 80% of cyber security breaches were due to human error and **20%** involved attacks on desktop and laptops [1]. Whether this is accidental (bugs and misconfigurations) or on purpose (attempting to relax or bypass organization security settings), it increases the risk of a breach. Per Trend-Micro, "If an attacker successfully compromises your build server, code repository, or developer workstations, they can reside in your environment for significantly longer." [2]

We've seen this play out in the real-world with recent supply chain attacks with malicious NPM packages, in which attackers create malicious packages that resemble legitimate ones (both in name and functionality) which developers then download and run in their workstations. These packages then install malware on the host machine to exfiltrate sensitive data, move laterally within the company's network, etc. [7] These aren't isolated events, in 2021, we witnessed supply chain attacks grow by over 300% [3].

These breaches show that security is needed at every step of the development lifecycle, starting with the developer's environment (e.g., laptops, virtual desktops, etc.).

This document describes some security risks in developer environments that use containers (e.g., a developer's laptop with a container engine installed), and how Docker Desktop can help to better secure them.

Security Risks in Container Development Tools

Container development tools (such as Docker Desktop and similar) enable users to package and run software inside containers, prior to deploying these in CI and production environments.

While running software inside containers is generally more secure than running it natively on a host (due to the container's reduced attack surface and host isolation), container development tools still need to be properly secured to reduce the risk of security breaches in the developer's environment.

We've identified five main threats associated with these tools:

1. Malware Attacks
2. Supply Chain Attacks
3. Local Admin Rights

4. Misconfigurations
5. Insider Threats

Malware Attacks

Malware attacks continue to gain prevalence and account for over 30% of cyber security breaches in 2021 [1].

In the context of container dev environments (e.g., a developer's laptop where Docker is installed), malware attacks most commonly occur when developers inadvertently download a malicious container image in their local machine and run it, or when a developer exposes the container engine API on the network without protection.

TechTarget reports that over one thousand malicious NPM packages have been detected, while Trend-Micro reports malicious container images posing as popular Alpine images and Aqua reports a similar thing for OpenJDK and Golang base images. In addition, SysDig also reports on the perils of exposing the container engine API on the network without authentication.

Mitigating malware in containers is important because not only can it execute harmful activities within the container (e.g., cryptomining), but can also try to gain access to the host's file system and network. Such access can occur when developers mount host files into the container (a common practice in dev environments) or when they run the container with elevated privileges.

Once the malware gains access to the host filesystem and network, it can exfiltrate sensitive data and credentials, use those to access private company repositories, attack other hosts on the same network, and more.

In some cases it's even possible for the malicious container to get access to the container engine. This can happen when developers expose the engine's API on the local host without protection (i.e., without TLS), or when they mount directories such as "/var", "/bin", "/run" into the container. This gives the malicious container access to the container engine's socket which it can use to pull private images, inspect them, create corrupt images, push them to repositories, etc.

There are several ways to mitigate malware threats to containers in dev environments:

- Ensuring container images come from trusted sources and are verified.
- Isolating containers from the developer's host machine.
- Isolating containers from the container engine.
- Isolating containers from each other.
- Restricting what host files can be shared with containers.
- Restricting host network access to containers.
- Never exposing the container engine API on the network without authentication.
- Ensuring the container dev environment is up to date.

Ideally, every container dev tool should offer all of the above. Taking it one step further, the tool should also allow IT admins to pick and choose which of these to enforce on developers' workstations, but do so in a way that does not impact developer experience and productivity.

Supply Chain Attacks

Recent supply chain attacks such as Log4shell, SolarWinds, and CodeCov have had a massive impact on organizations worldwide, underscoring the increasing risk posed by an insecure supply chain. In fact, supply chain attacks grew by over 300% in 2021. This is likely due to the economies of scale for attackers: a single vulnerability in a key supply chain software component can result in thousands of impacted systems.

300%

Increase in supply chain attacks in 2021

In the context of container dev environments, supply chain attacks can occur in a variety of ways:

- Developers inadvertently pull a malicious package into a container (e.g., malicious NPM packages [7]).
- Developers use a compromised or vulnerable container image as a layer in other images they create.
- Malware infects a developer's container dev environment (see prior section) and covertly inserts vulnerabilities into artifacts created by the developer. For example, if malware gains control over the container engine, it can push malicious images to the company's repository or even corrupt image builds created by the developer.

Attackers are increasingly using techniques such as typosquatting [5] and dependency confusion [10] of package and container image names to trick developers into downloading malicious containers or packages..

The scalability of supply chain attacks means they can put affected organizations in serious financial and regulatory jeopardy. Therefore mitigating such attacks in developer environments is critical. This includes ensuring only trusted container workloads are run and, in case an attack still does occur, limiting the blast radius. Here's what you can do to achieve this:

- Ensure developers can only pull images from trusted sources / registries.
- Constantly scan the container images used and produced by developers.
- Give containers least privilege.
- Avoid storing secrets inside container images.
- Mount as few host directories as possible into the containers, particularly those that contain sensitive data (e.g., avoid mounting the user's $HOME directory since it contains SSH keys).
- Mount directories with read-only permissions (as opposed to read-write).
- Harden container isolation in the dev environment (to prevent container access to the Linux host and the container engine).

Also, IT admins should have the ability to set and enforce security settings in developer workstations, such as the container registries and images that developers can access.

Local Admin Rights

Giving developers local admin rights on their machines might make them happy, but if their workstation is not configured securely per company policies, this can quickly lead to attackers compromising these machines and from there, the organization's network. With local admin rights, developers can relax firewall rules, turn off anti-malware software, disable updates, and install untrusted software.

Ideally there should be a balance where developers are able to perform any legitimate action without local admin rights on their workstation, and IT/systems admins have the ability to apply and enforce security settings on the developer's workstation. The goal is to secure without impacting developer experience or productivity.

These tools should allow developers to do their job (e.g., launch containers, build images, upload images) using standard local machine accounts, while at the same time giving admins the ability to configure and enforce security controls, such that they can't be relaxed without authorization.

Misconfiguration

Container dev tools give developers a wide range of configuration options, even without local admin rights. Misconfiguring these can result in non-compliance of policy and increased security threats. In 2021, ~7% of cyber security breaches were due to misconfiguration [1].

Misconfigurations happen because most container dev tools give developers full control over the configurations and settings of the tool. Even if an IT admin initially configures the tool's settings per the company's security policy, a developer can always modify these settings at will. In other words, misconfigurations occur when there is no centralized or mandatory organizational control over the tool's security settings.

For example, within Docker Desktop developers can change a large number of settings such as the network proxy config, host file sharing, host resource usage, Docker engine config, software updates, etc. This gives developers great flexibility.

However, a few of these settings can increase security risks and in some organizations that risk outweighs the benefit. An incorrect configuration (e.g., disabling the network proxy or disabling software updates) may result in developers opening the door for attacks on their workstation and compromising the entire organization.

Organizations that operate with strict security requirements or at scale need a mechanism that allows IT admins to configure and enforce some security settings in container dev tools, but leave other settings at the discretion of the developers to prevent negative impacts to developer experience and productivity. Developers should be able to configure their container development environment in the way that best suits them, while at the same time being restricted from relaxing important security settings set by IT admins.

Insider Threats

Threats don't always come from external parties. Malicious insiders are a growing and real concern with 20% of all breaches in 2021 caused by internal actors [1]. Development tools are a prime target for these malicious insiders because they can be leveraged to perform malicious activities disguised as legitimate ones given what these tools can do.

Once they've compromised your systems, these insiders will try to bypass company security policies to exfiltrate confidential company data, run disallowed software, download malware, or even intentionally introduce vulnerabilities.

For container development tools in particular, this could mean pulling malicious images, running them in the developer's environment (which may go undetected by anti-malware software since these tools typically run containers inside a virtual machine), pushing corrupt container images to an organization's repositories, etc.

Securing container dev tools against insider threats requires that IT/system admins be able to enforce organizational security policies on these tools so that these can't be used to perform hostile activities.

Threat Paths

Figure 1 below shows possible attack paths for the threats identified above. These apply to all container dev tools. They assume, however, that containers inside a Linux VM (as Docker Desktop and other tools do in order to run containers on Mac or Windows hosts).

The diagram is read from bottom-to-top, and the white boxes represent the threat action while the blue boxes represent the end result.

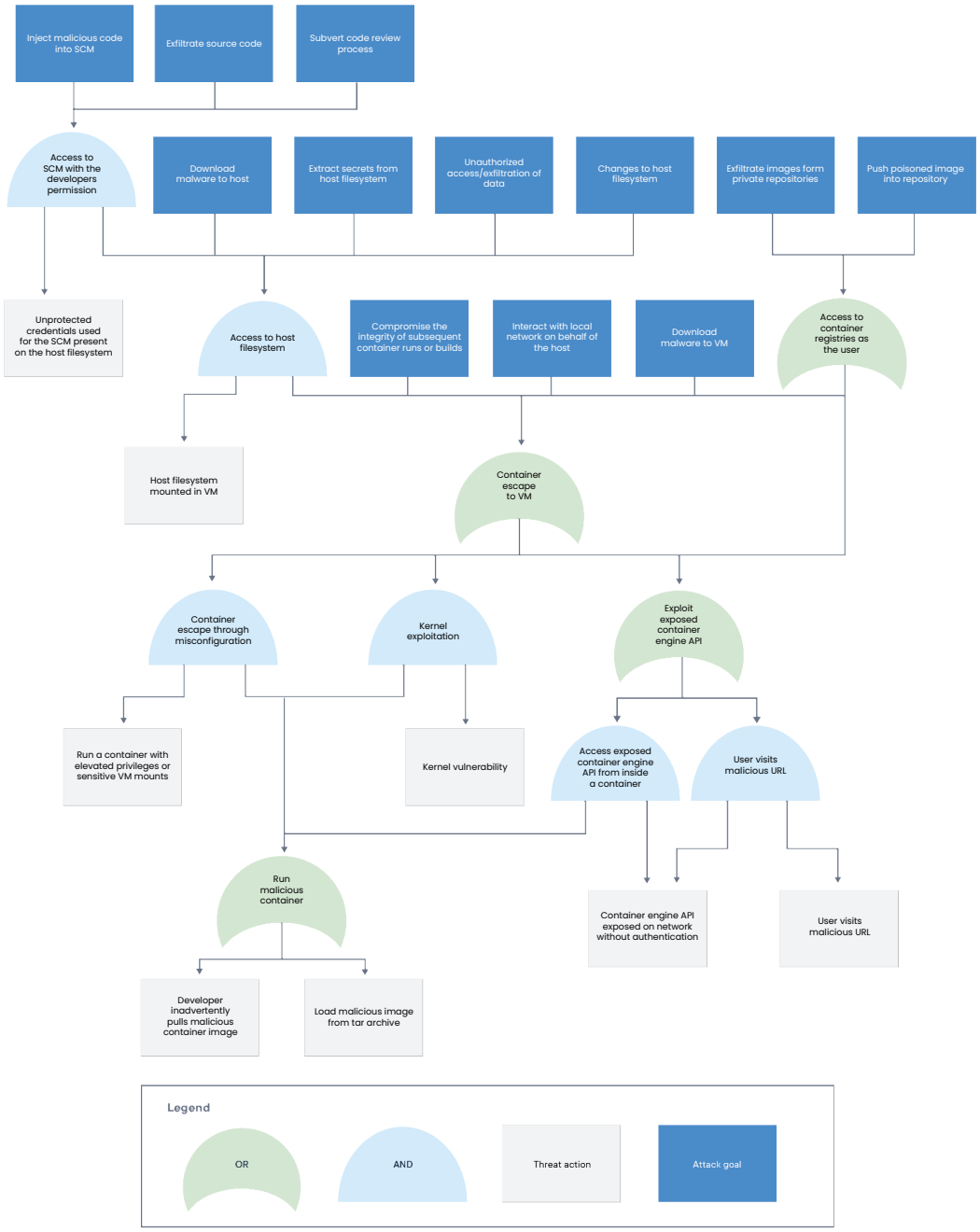

Figure 1. Container Dev Tool Threats Paths

Docker Desktop Security Features and Limitations

Docker Desktop includes important baseline security features to protect against most of the threats described in the prior section. This section outlines these baseline security features and their limitations. In the following section we introduce a new security feature created by Docker to overcome these limitations.

Baseline security features include:

- Allowing regular users to run containers (no admin rights needed).
- Running all containers in a Linux Virtual Machine (VM) to isolate them from the underlying Mac/Windows/Linux host.
- Preventing containers from accessing host files not owned by the user.
- Offering configurations for network proxies, registries, tool updates, host file sharing, and more.
- Tight integration with Docker Hub for access to private repositories, automated image scanning, Docker Official Images, Docker Verified Publisher images, and more.

These baseline security features are all available in Docker Desktop and most other free container development tools. However, they have some important limitations (and these apply to other container development tools too):

- Security controls are discretionary. They are built so that individual developers can modify them and there is no centralized / mandatory control over them. This opens the door for misconfiguration or insider threats when used in an enterprise setting. An unaware or careless developer, or a malicious insider, could relax these settings and put the organization at risk.
- Container-to-host isolation is strong by virtue of running the containers inside a Linux VM. However, within the Linux VM, container isolation can easily be relaxed or bypassed (see below), opening the door for malware or supply chain attacks to subvert the VM and the container engine within it.

For example, if a developer inadvertently runs a malicious container image with elevated privileges (e.g., “docker run --privileged”) or shares namespaces (e.g., “docker run –pid=host”), the malicious container now has root access to the Linux kernel in the VM. The container can use this to subvert security controls within the VM.

Even without elevated privileges, if the developer exposes sensitive VM directories to the malicious container (e.g., “docker run -v /var:/mnt” or “docker run -v /bin:/mnt”), the malware can also install itself in the VM or take control of the container engine within it.

Figure 2 illustrates this.

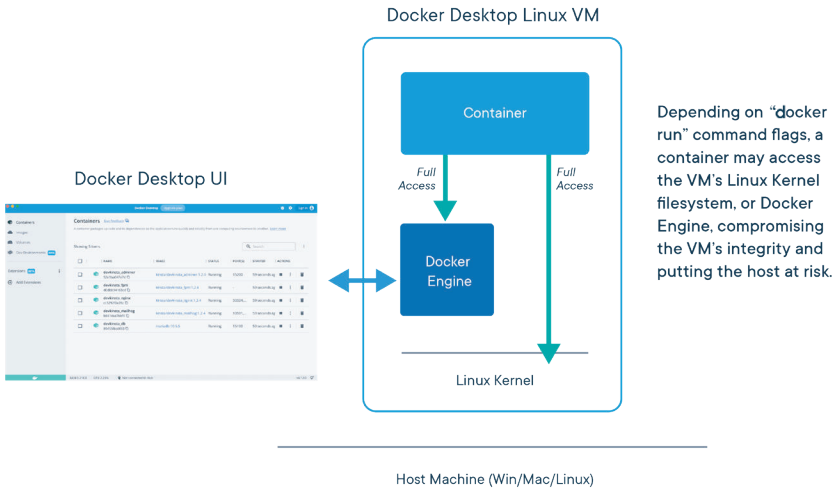

Figure 2. Possible ways to compromise the Docker Desktop Linux VM

Once the VM is compromised, it introduces risks to the confidentiality and integrity of assets within the VM (which may include host files exposed in the VM). Some examples of this are:

- Accessing, modifying, and exfiltrating improperly stored credentials, source code, or other sensitive information from the host machine's filesystem. This is available in the VM even without being explicitly shared with the container.
- Taking control of Docker Engine (whose socket is available inside the VM at "/var/run/docker.sock") and leveraging it to pull more malware, build and push malicious images, etc.
- Modifying the binaries for Docker Engine or other programs inside the VM. This compromises the integrity of subsequent runs or builds, leading to supply chain attacks within the organization.
- Pushing code changes to GitHub and subverting code review process in case of improper authentication. For example, if a user shares their host's $HOME directory, a malicious container that gained access to the Linux VM could steal the SSH private key and compromise access to GitHub and similar services.
- Utilizing the interfaces between the container tool and the VM to attack the host or modify settings stored in the VM without the developer's knowledge.
- Attempting to take over the host by accessing its filesystem (e.g. editing the user's shell profile, performing a dynamic link library (DLL) sideloading attack by writing a malicious DLL, etc.)
- Subverting other security controls, network/proxy configurations, or observability features enforced from within the VM (e.g. Registry Access Management, proxy configuration).
- Interacting with the local network on the developer's behalf in order to perform reconnaissance and identify further weaknesses, potentially leading to lateral movement via attacking other hosts on the network.

Malware that installs itself inside the container tool VM will likely go undetected by anti-malware soft-

ware on the host. This is because the VM acts like a black box, keeping anti-malware software on the host from seeing what's happening inside of it.

In addition, malware in the VM would survive a restart because the VM's storage remains untouched. And if the user disables updates, the malware could live in the machine for long periods of time, undetected.

These security limitations are true for Docker Desktop as well as all other container development platforms in the market that don't have enterprise-level security and IT/system administrators features.

Even "rootless" container runtimes (where the container engine runs without root privileges) are not immune to this because:

- They don't prevent users from tampering with security settings (e.g., developers can trivially disable the rootless feature).
- They don't offer proper isolation between the containers and the container engine inside the VM. Rootless runtimes normally run the containers and container engine in the same Linux user-namespace, so while they help isolate the container from the Linux kernel, they don't prevent containers from accessing the container engine, a prime target for attackers.

In companies with stringent security requirements, this level of security is not sufficient. These companies often need a mechanism for IT/system admins to enforce security settings on the container development platforms (e.g., proxies, registries, Docker Engine configs) with stronger guarantees that won't be easily bypassed by developers or malicious containers.

Hardened Docker Desktop: Stronger Security for Enterprises

The baseline Docker Desktop security features described in the prior section may not be sufficient for enterprises.

To address this, Docker Desktop 4.13 introduces "Hardened Docker Desktop" (HDD), a new set of security features for Docker Desktop that overcomes these limitations, and does so in a way that is transparent to developers and doesn't impact their productivity, velocity, and experience.

Hardened Docker Desktop gives IT admins a simple, powerful, and centralized way to secure Docker Desktop deployments across the entire company. It also hardens container isolation to make it much more difficult to breach the Docker Desktop Linux VM and the underlying host.

With HDD, developers can continue to work with Docker Desktop as usual (including running privileged containers) knowing that their Docker Desktop deployment is running much more securely.

This ultimately improves the company's security posture and protects against security vulnerabilities introduced by users or malicious code. Hardened Docker Desktop is only available for Docker Business customers.

What's Included?

Hardened Docker Desktop includes the following features:

- Settings Management
- Enhanced Container Isolation (ECI)
- Registry Access Management
- Image Access Management

Settings Management

Settings Management allows IT/system admins to preset and "lock" several Docker Desktop configurations on a developer's machine. Once set, admins can have confidence that those configurations will not be altered by their developers. This is because only the IT/system admin will have admin rights into the developer's workstation. These configurations include:

- Proxy
- Container registries
- Host files shared with containers
- Docker Engine configurations
- Kubernetes settings
- Enhanced Container Isolation
- Software updates
- More planned in future Docker Desktop releases.

Administrators can get started by creating a file called "admin-settings.json" which contains the desired set of settings the admin wishes to preset and lock. Figure 3 below shows a simple example of this file (many more settings are possible).

```
{
    "configurationFileVersion": 2,
    "exposeDockerAPIonTCP2375": {
        "locked": true,
        "value": false
    },
    "enhancedContainerIsolation": {
        "locked": true,
        "value": true
    },
    "disableUpdate": {
      "locked": true,
      "value": false
    }
}
```

Figure 3. Sample "admin-settings.json" file used for Settings Management.

This file resides in the computer where Docker Desktop is installed, in a directory that is only accessible with local admin rights on the machine. When Docker Desktop starts, it looks for that file and locks the appropriate settings. Developers will see those locked settings in the Docker Desktop GUI and won't be able to modify them, as shown in Figure 4.

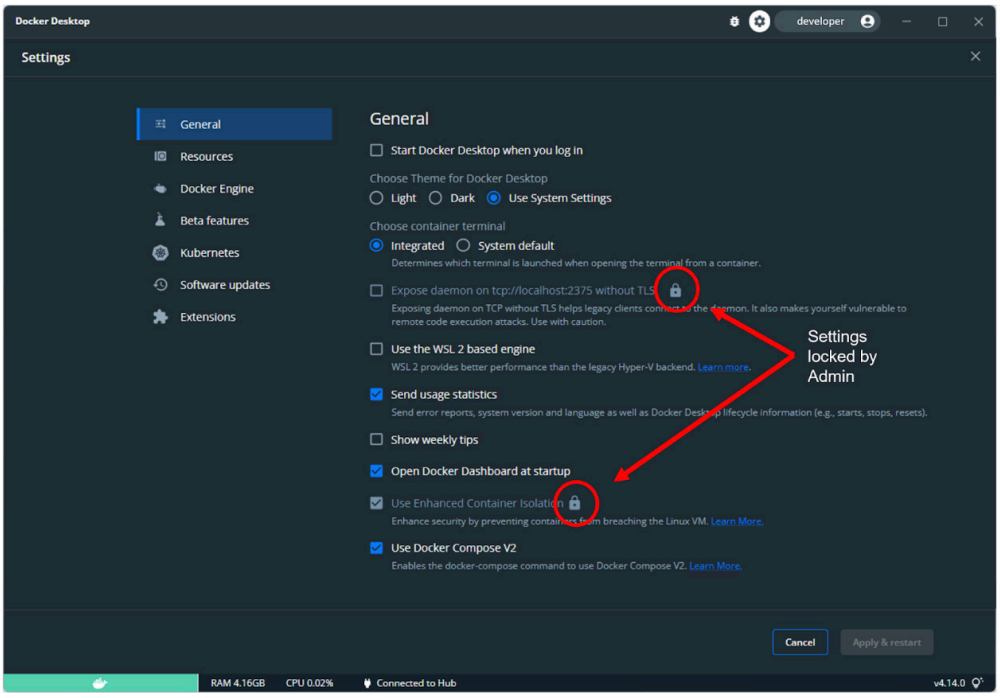

Figure 4. Docker Desktop UI showing settings locked by Admins.

Settings Management enables mandatory access control (MAC) on Docker Desktop settings (as opposed to the baseline discretionary access control). Read more in the Settings Management documentation.

Enhanced Container Isolation (ECI)

Enhanced Container Isolation (ECI) hardens the security of containers inside the Docker Desktop Linux VM by running them unprivileged (i.e., root user in the container maps to an unprivileged user inside the VM) to prevent them from taking control of the VM. Even containers launched with the "--privileged" flag are protected this way.

This significantly hardens container isolation and prevents containers from bypassing Docker Desktop registry configurations, accessing sensitive data or secrets stored in the VM, accessing the Docker Engine, peeking into other containers, or attacking the host machine. In addition, several CVEs related to container escapes (e.g. CVE 2019-5736, CVE 2022-0492, CVE 2021-30465) are blocked by ECI.

ECI uses several advanced and unique techniques to do this:

- Running all containers unprivileged (Linux user namespace).
- Restricting users from exposing directories in the Linux VM inside the container.

- Preventing backdoor access to the Docker Desktop VM (e.g., debug console).
- Vetting sensitive system calls such “mount” in the container to ensure they don’t result in container breakouts.
- Emulating portions of the procfs and sysfs filesystems inside the container to further isolate the container from the underlying Linux kernel.

Figure 5 below illustrates this.

ECI is a critical component of Hardened Docker Desktop, as it ensures the Docker Desktop Linux VM can’t be easily subverted by Docker Desktop users or malicious containers. ECI can be enabled and enforced by IT/system admins via the admin-settings.json file (see section Settings Management above).

ECI is unique to Docker Desktop, and goes well beyond just using the Linux user namespace (as rootless features in other container runtimes do). Rather, it uses the Linux user namespace as a baseline for isolation but complements this with the advanced techniques listed above to harden container isolation. Read more in the Docker Desktop documentation.

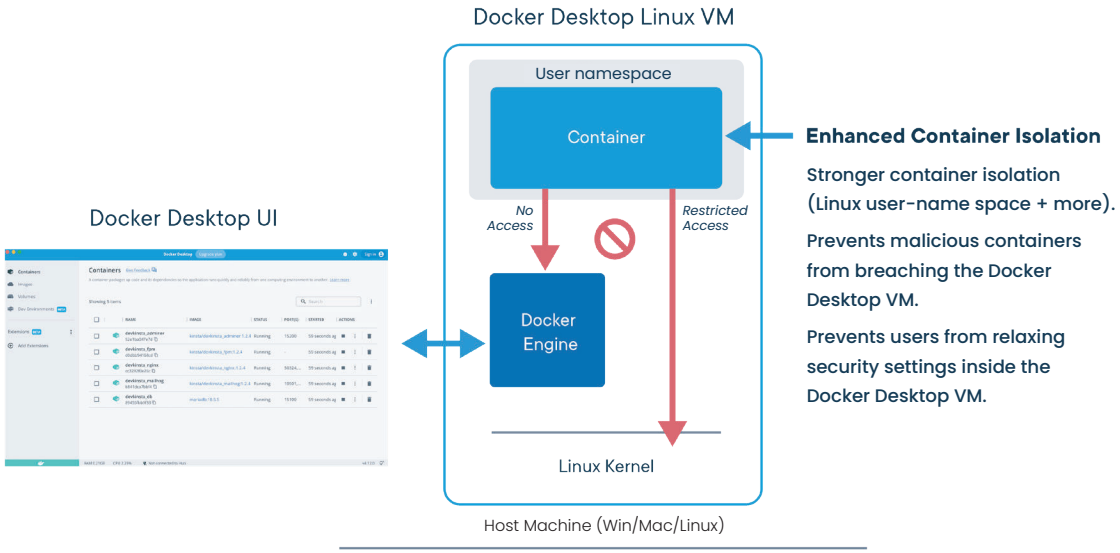

Figure 5. Enhanced Container Isolation makes it much harder for containers to breach the Docker Desktop Linux VM.

Registry Access Management

Registry Access Management allows admins to control which online registries Docker Desktop can pull or push artifacts from (e.g., container images).

Registry Access Management plays a key role in preventing malware and supply chain attacks (such as those recently discovered by Aqua [5] and SysDig [6]) since it ensures developers can only pull container images from authorized, vetted registries. Without this feature, developers may pull images from untrusted registries at their discretion, putting the security of your company at risk.

Registry Access Management is configured by administrators via Docker Hub. Refer to the Docker Desktop documentation for more information.

Image Access Management

Image Access Management allows organizations to control which types of images their developers can pull from Docker Hub (e.g., Docker Official Images, Docker Verified Publisher Images, Community images).

For example, a developer who is building a new containerized application could accidentally use an un-trusted community image as a component of their application. This image could be malicious and pose a security risk to the company. Using Image Access Management, the organization owner ensures that the developer can only access trusted content like Docker Official Images, Docker Verified Publisher Images, or the organization's own images, mitigating such a risk.

Hardened Docker Desktop Threat Mitigation

This section outlines how Hardened Docker Desktop (HDD) addresses the security threats described in the “Security Risks” section above.

The features of Hardened Desktop (Registry Access Management, Image Access Management, Settings Management, and Enhanced Container Isolation (ECI)) operate independently of each other, but when used together can create a defense-in-depth approach to securing developer workstations.

Defense-in-depth means that these mechanisms defend against attacks at different functional layers (e.g., when configuring Docker Desktop, when pulling container images, when running container images, etc.), as described below.

Figure 6 illustrates how Hardened Desktop mitigates threats.

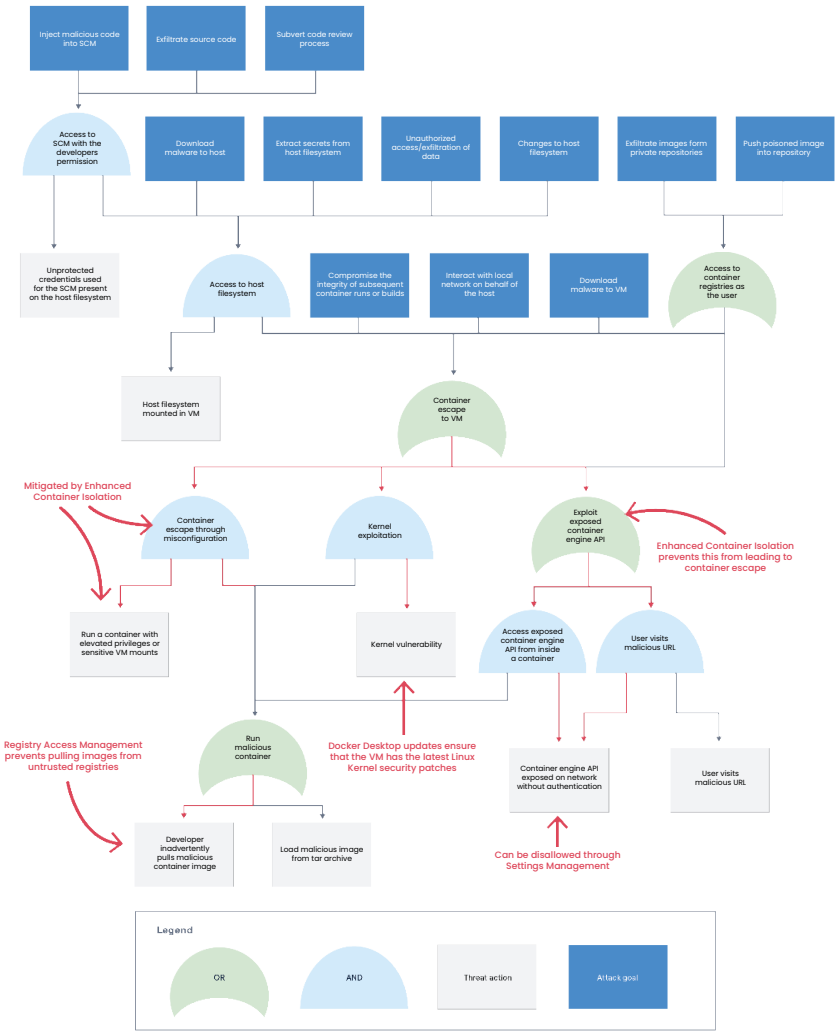

Figure 6. Hardened Desktop Threat Mitigation

Mitigating Malware with HDD

IT administrators can use Registry Access Management and Image Access Management to restrict which container registries and image types developers can access for pushing and pulling container images. This significantly reduces the probability that container images will have malicious payloads. Developers can't bypass this setting.

In addition, with ECI enabled, even if a container with a malicious payload runs, it's much harder for that container to breach the Docker Desktop Linux VM (and from there the host) because the container runs without root privileges inside a Linux user namespace. The container is also restricted from accessing any files inside the Linux VM or from accessing the Docker Engine. Even if users run containers with elevated privileges (e.g., "--privileged" or "--cap-add=ALL"), the container is prevented from breaching the Linux VM.

Securing against Supply Chain Attacks with HDD

Registry Access Management and Image Access Management restricts the registries and image types that developers can pull from. This reduces the probability of supply chain attacks caused by corrupt images.

If a malicious package is inadvertently installed in the container, ECI prevents the malicious package from running as root inside the Docker Desktop Linux VM and from accessing the Docker Engine within it, creating another layer of defense.

Finally, Settings Management allows administrators to lock settings and prevent users from changing them. This includes locking Docker Desktop's proxy settings and ensuring that ECI is always enabled.

Avoiding Misconfigurations with HDD

IT admins can use Settings Management to configure and lock settings on local Docker Desktop installations. That includes fixing the proxy settings, enabling ECI, preventing users from exposing the unprotected Docker API on the host's network, enabling software updates, etc.

IT Admins can choose to lock whatever settings are appropriate according to company policy, and developers can't modify them.

In addition to Settings Management, IT admins can leverage Registry Access Management and Image Access Management to configure which container registries and image types developers can access. This prevents developers from misconfiguring these important settings.

Addressing Insider threats with HDD

IT admins can use Settings Management to lock Docker Desktop settings and prevent malicious insiders from changing these to bypass organization's security policies.

In addition, ECI prevents malicious insiders from leveraging containers to gain control of the Docker Desktop Linux VM. The VM's debug console is also disabled, closing another entry route otherwise available to malicious insiders.

The table below summarizes the threats and threat mitigation features.

Category	Threat / Vulnerability	Mitigation Tactic	Other Container Dev Tools	Docker Desktop	Hardened Docker Desktop
Container Malware	Host Attack	Run containers in a Linux VM, config host file sharing.	✓	✓	✓
	Linux VM Attack	Prevent containers from accessing Linux VM.			✓ (ECI)
	Container Engine Attack	Prevent containers from accessing the container engine.			✓ (ECI)
	Cross-Container Attack	Strong cross-container isolation.			✓ (ECI)
Supply Chain Attacks	Unrestricted container registry access	Registry Access Management	✓ (easy for developers to bypass)	✓ (easy for developers to bypass)	✓ (can't be bypassed)
	Unrestricted image type access	Image Access Management			✓
Misconfiguration	Developer relaxes security setting in container tool GUI	Settings Management			✓
	Developer relaxes security setting inside the Linux VM	Prevent developer access to the Linux VM (via containers or directly)			✓
Host Admin Rights	Requiring host admin rights to run containers	Allow regular users to run containers	✓	✓	✓

Hardened Docker Desktop Roadmap

In addition to what's currently included in HDD, Docker plans on building more features that will further secure developer environments.

Roadmap Feature	Description
Enabling Enhanced Container Isolation on Docker Desktop Kubernetes Pods.	Better secures the Kubernetes dev cluster available in Docker Desktop.
Enabling Enhanced Container Isolation on Docker Desktop Extensions.	Better secures Extension containers.
Enabling Enhanced Container Isolation on Docker Desktop Dev Environments.	Better secures Dev Environment containers.
Supporting Hardened Desktop on Windows Subsystem for Linux 2 (WSL2)	Add support for Hardened Desktop on WSL2, with the caveat that WSL2 is by design less secure than Hyper-V.

Conclusion

Cyber security threats continue to grow and attacks increasingly target developer environments. Such threats, if not mitigated, can result in a breach, bringing severe financial, legal, and reputational consequences to companies.

As attackers evolve their tactics, techniques, and procedures, companies need to do the same with their defenses. Investing in solutions that can proactively fend off these attacks while giving developers the flexibility to continue using the tools they love, is a must.

Container development platforms such as Docker Desktop and others present unique security challenges since developers use them to constantly pull container images from the internet and run them in local environments.

While Docker Desktop provides strong baseline security features (e.g., running containers inside a Linux VM to isolate them from the underlying host), enterprises that operate at scale or under strictly regulated environments require more.

Hardened Docker Desktop, included in the Docker Business subscription, is an enterprise-ready solution that hardens security in your developer's environments, without impacting your developer productivity and experience.

All of the features in Hardened Docker Desktop - Settings Management, Enhanced Container Isolation, Registry Access Management, and Image Access Management - work in conjunction to secure the developer workstation from the start. So you will have peace of mind that your developers are staying compliant and protecting your company from threats, while at the same time enjoying the productivity and innovation made possible by Docker.

Learn More

Hardened Docker Desktop Online Documentation.

References

[1] Verizon Data Breach Investigation Report, 2022
[2] https://www.trendmicro.com/en_us/what-is/container-security.html
[3] https://www.aquasec.com/cloud-native-academy/supply-chain-security/software-supply-chain-attacks/
[4] https://www.trendmicro.com/vinfo/it/security/news/virtualization-and-cloud/malicious-docker-hub-container-images-cryptocurrency-mining
[5] https://blog.aquasec.com/supply-chain-threats-using-container-images
[6] https://sysdig.com/blog/triaging-malicious-docker-container/
[7] https://www.bleepingcomputer.com/news/security/npm-supply-chain-attack-impacts-hundreds-of-websites-and-apps/
[8] https://www.ibm.com/reports/data-breach
[9] https://github.com/OWASP/Docker-Security
[10] https://snyk.io/blog/detect-prevent-dependency-confusion-attacks-npm-supply-chain-security