Secure Windows containers

Article • 04/29/2024 • 10 contributors

Feedback

In this article

> Applies to: Windows Server 2022, Windows Server 2019, Windows Server 2016

Containers lend their reduced image size to the fact that they can rely on the host to provide limited access to various resources. If the container knows that the host will be able to provide the functionality needed to perform a specific set of actions, then the container does not need to include the relevant software in its base image. The extent of resource sharing that occurs, however, can have a significant impact on both the performance and security of the container. If too many resources are shared, then the application may as well just run as a process on the host. If the resources are shared too little, then the container becomes indistinguishable from a VM. Both configurations are applicable to many scenarios, but most people using containers generally opt for something in the middle.

The security of a Windows container is derived from the degree of sharing that occurs with its host. The security boundary of a container is constituted by the isolation mechanisms that separate the container from the host. Most importantly, these isolation mechanisms define which processes in the container can interact with the host. **If the design of a container allows for an elevated (admin) process to interact with the host, then Microsoft does not consider this container to have a robust security boundary.**

Windows Server containers vs. Linux containers

Process-isolated Windows Server containers and Linux containers share similar degrees of isolation. For both container types, the developer must assume any attack that can be performed via an elevated process on the host can also be performed via an elevated process in the container. Both operate via the primitive capabilities offered by their respective host kernels. The container images are built containing user mode binaries that utilize the APIs provided by the host kernel. The host kernel provides the same resource isolation and management capabilities to each container running in user space. If the kernel is compromised, then so is every container sharing that kernel.

The fundamental design of Linux and Windows server containers trades security for flexibility. Windows Server and Linux containers are built on top of the primitive components provided by the OS. Doing so provides the flexibility for sharing resources and namespaces between containers, but it adds additional complexity that opens the door for exploitation. Broadly stated, **we do not consider the kernel to be a sufficient security boundary for hostile multi-tenant workloads.**

Kernel isolation with hypervisor-isolated containers

Hypervisor-isolated containers provide a higher degree of isolation than process-isolated Windows Server or Linux containers and are considered robust security boundary. Hypervisor-isolated containers consist of a Windows Server container wrapped in an ultralight VM, and then run alongside the host OS via Microsoft's hypervisor. The hypervisor provides hardware-level isolation that includes a highly robust security boundary between the host and other containers. Even if an exploit were to escape the Windows Server container, it would be contained within the hypervisor-isolated VM.

Neither Windows Server containers or Linux containers provide what Microsoft considers a robust security boundary and should not be used in hostile multi-tenant scenarios. A container must be confined to a dedicated VM to prevent a rogue container process from interacting with the host or other tenants. Hypervisor isolation enables this degree of separation while also offering several performance gains over traditional VMs. Therefore, it is highly recommended that in hostile multi-tenant scenarios, hypervisor-isolated containers should be the container of choice.

Container security servicing criteria

Microsoft is committed to patching all exploits and escapes that break the established isolation boundary of any Windows container type. However, only exploits that break a security boundary are serviced via the Microsoft Security Response Center (MSRC) process. **Only hypervisor-isolated containers provide a security boundary, and process-isolated containers do not.** The only way to generate a bug for a process-isolated container escape is if a non-admin process can gain access to the host. If an exploit uses an admin process to escape the container, then Microsoft considers it to be a non-security bug and will patch it per the normal servicing process. If an exploit uses a non-admin process to perform an action that breaches an security boundary, then Microsoft will investigate it per the established security servicing criteria.

What makes a multi-tenant workload hostile?

Multi-tenant environments exist when multiple workloads are operating on shared infrastructure and resources. If one or more workloads running on an infrastructure cannot be trusted, then the multi-tenant environment is considered hostile. Both Windows Server and Linux containers share the host kernel, so any exploit performed on a single container can impact all other workloads running on the shared infrastructure.

You can take steps to reduce the chance that an exploit will occur, for example, by using pod security policies, AppArmor, and role-based access control (RBAC), but they do not provide guaranteed protection against an attacker. We recommend you follow our best practices for cluster isolation for any multi-tenant scenario.

When to use ContainerAdmin and ContainerUser user accounts

Windows Server containers offer two default user accounts, ContainerUser and ContainerAdministrator, each with their own specific purpose. The ContainerAdministrator account enables you to use the container for administrative purposes: installing services and software (such as enabling IIS within a container), making configuration changes, and creating new accounts. These tasks are important for a number of IT scenarios that may be running in custom, on-premises deployment environments.

The ContainerUser account exists for all other scenarios where administrator privileges in Windows are not needed. For example, in Kubernetes if the user is ContainerAdministrator and hostvolumes are permitted to be mounted into the pod then the user could mount the .ssh folder and add a public key. As ContainerUser this example would not be possible. It is a restricted user account designed purely for applications which do not need to interact with the host. **We strongly recommended that when deploying a Windows server container to any multi-tenant environment that your application runs via the ContainerUser account.** In a multi-tenant environment, it is always best to follow the principle of least privilege because if an attacker compromises your workload, then the shared resource and neighboring workloads have a lower chance of being impacted as well. Running containers with the ContainerUser account greatly reduces the chance of the environment being compromised as a whole. However, this still does not provide a robust security boundary, so when security is a concern you should use hypervisor-isolated containers.

To change the user account, you can use the USER statement on your dockerfile:

Dockerfile — Copy

```
USER ContainerUser
```

Alternatively, you can create a new user:

Dockerfile — Copy

```
RUN net user username '<password>' /ADD
USER username
```

Windows services

Microsoft Windows services, formerly known as NT services, enable you to create long-running executable applications that run in their own Windows sessions. These services can be automatically started when the operating system starts, can be paused and restarted, and do not show any user interface. You can also run services in the security context of a specific user account that is different from the logged-on user or the default computer account.

When containerizing and securing a workload that runs as a Windows service there are a few additional considerations to be aware of. First, the `ENTRYPOINT` of the container is not going to be the workload since the service runs as a background process, typically the `ENTRYPOINT` will be a tool like service monitor) or log monitor). Second, the security account that the workload operates under will be configured by the service not by the USER directive in the dockerfile. You can check what account the service will run under by running `Get-WmiObject win32_service -filter "name='<servicename>'" | Format-List StartName`.

For example, when hosting an IIS web application using the ASP.NET (Microsoft Artifact Registry) image the `ENTRYPOINT` of the container is `"C:\\ServiceMonitor.exe", "w3svc"`. This tool can be used to configure the IIS service and then monitors the service to ensure that it remains running and exits, thus stopping the container, if the service stops for any reason. By default, the IIS service and thus the web application run under a low privilege account within the container, but the service monitor tool requires administrative privileges to configure and monitor the service.

Feedback

Was this page helpful? Yes No

Additional resources

Training

Module
Run containers on Windows Server - Training
Run containers on Windows Server

Certification
Microsoft Certified: Windows Server Hybrid Administrator Associate - Certifications
As a Windows Server hybrid administrator, you integrate Windows Server environments with Azure services and manage Windows Server in on-premises networks.

Documentation

Windows container base images
Overview of the Windows container base images and when to use them.

Windows container version compatibility
Version compatibility for containers built from different versions of Windows Server and Windows.

Windows Containers FAQ
Windows Server containers FAQ

Show 5 more