Alpine Linux - Bug #7093

Issue with getmntent_r with overlay/overlay2 with /proc/mounts entries >=1024 bytes

04/05/2017 04:52 PM - Dimitrios Liappis

Status:	Closed	**Start date:**	04/05/2017
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	3.6.1		
Affected versions:	3.5.2	**Security IDs:**	

Description

While investigating an issue with Java and how JNI is using getMountEntries() and it's getmntent JNI dispatcher we came across a discrepancy between how getmntent_r() behaves in musl and glibc on the following conditions:

- docker storage driver is overlay or overlay2
- there are enough layers so that the overlay / entry in proc mounts in longer than 1024 characters

Note that this is separate from the older Alpine bug#5703 that deals with the 256byte fixed limit.

Here are the reproduction steps:

1. Ensure `docker info` reports overlay/overlay2. This is the default, e.g. on Fedora, Docker for Mac and Docker for Windows.

2. Create the following files in your dir:

```
cat >docker-compose.yml <<EOF
version: '2'

services:
  mytester:
    build: .
    image: testgetmntent:alpine
EOF


cat >Dockerfile <<EOF
FROM alpine:latest
RUN apk update && apk add bash gcc musl-dev && rm -rf /var/cache/apk/*
RUN echo test 0 > /tmp/test0
RUN echo test 1 > /tmp/test1
RUN echo test 2 > /tmp/test2
RUN echo test 3 > /tmp/test3
RUN echo test 4 > /tmp/test4
RUN echo test 5 > /tmp/test5
RUN echo test 6 > /tmp/test6
RUN echo test 7 > /tmp/test7
RUN echo test 8 > /tmp/test8
RUN echo test 9 > /tmp/test9
RUN echo test 10 > /tmp/test10
RUN echo test 11 > /tmp/test11
EOF
```

3. Build and run the resulting image with:

```
docker-compose build --pull && docker-compose run mytester /bin/bash


4. In the shell, create the following C program to test getmntent_r:


cat >getmntent_r_tester.c <<EOF
```

```
#include <stdio.h>
#include <mntent.h>

void print_mount(const struct mntent *fs);

int main(int argc, char **argv)
{
    FILE *fp;
    struct mntent *fs;
    struct mntent ent;
    char buf[1024];
    int buflen = sizeof(buf);

    fp = setmntent("/proc/mounts", "r");    /* read only */
    if (fp == NULL) {
        printf("Error calling setmntent\n");
        return(1);
    }
    while ((fs = getmntent_r(fp, &ent, (char*)&buf, buflen )) != NULL)
    print_mount(fs);

    endmntent(fp);
}

void print_mount(const struct mntent *fs)
{
    printf("%s %s %s %s %d %d\n",
        fs->mnt_fsname,
        fs->mnt_dir,
        fs->mnt_type,
        fs->mnt_opts,
        fs->mnt_freq,
        fs->mnt_passno);
}
EOF
```

5. Compile and run it. Notice there's no output.

```
bash-4.3# gcc getmntent_r_tester.c
bash-4.3# ./a.out
```

6. Notice that overlay / entry in /proc/mounts exceeds 1024bytes:

```
bash-4.3# grep overlay /proc/mounts | wc -c
1069
```

7. Exit the container, run docker-compose down.

8. Edit the Dockerfile and remove the last 2 RUN commands

9. Rebuild, rerun the image and recompile and rerun the C program, in steps 3-5:

```
bash-4.3# gcc getmntent_r_tester.c
bash-4.3# grep overlay /proc/mounts | wc -c
961
bash-4.3# ./a.out
overlay / overlay rw,seclabel,relatime,lowerdir=/var/lib/docker/overlay2/l/LVWNODYSVRXAQW4P5QXNVVX
DSS:/var/lib/docker/overlay2/l/RVAZXPKSLDGTV5QXZLI6HZAXXU:/var/lib/docker/overlay2/l/EJIOFNYRMEPTJ
4AID2PXUZNJOC:/var/lib/docker/overlay2/l/P4BG3VO46BQDHUXLXPZ3KDXAIR:/var/lib/docker/overlay2/l/G56
3LG47W5E6A4YPFZCTQ5ORUG:/var/lib/docker/overlay2/l/E2PLIGC66VLBLKENQA2GUKOZXO:/var/lib/docker/over
lay2/l/6C5UXCJ4COZVHKHDRDKGW7PKIZ:/var/lib/docker/overlay2/l/GY45BC44L45ZYGQB7XCUU3L4LU:/var/lib/d
```

```
ocker/overlay2/l/FF4KRJU7RWCA2GQ6OZV4OHLCXM:/var/lib/docker/overlay2/l/5SZT4SYLCF5LZAZCAJIA6YFB3K:
/var/lib/docker/overlay2/l/3SP6Y2WURVZ4L6IZD3WQ2T5ZRG:/var/lib/docker/overlay2/l/LZZGUAD7WB6NZAKG5
PSE4K7S6B:/var/lib/docker/overlay2/l/5VA5PMTBPJ4VRPX7HNGUVOYARI,upperdir=/var/lib/docker/overlay2/
9ff9f8c3a876cec942dbfa5324894d24d1fe4eb52700320e5e5fa3e7e34a137a/diff,workdir=/var/lib/docker/over
lay2/9ff9f8c3a876cec942dbfa5324894d24d1fe4eb52700320e5e5fa3e7e34a137a/work 0 0
<snip here>
```

10. Trying the same process with a different docker image base, based on glibc, (e.g. fedora:latest) works in all cases.

Associated revisions

Revision 189a2712 - 05/30/2017 03:58 PM - Natanael Copa

community/openjdk8: increase buffer size for getmntent_r

Java will only use 1024 byte buffer for parsing mounts. Unlike glibc
will musl return error when this is not big enough instead of truncating
it.

We solve it by allocating a much bigger buffer.

fixes #7093

We also build without precompiled headers, which does not work eith PIE.

Revision d0a7b324 - 06/13/2018 09:18 PM - Natanael Copa

community/openjdk8: increase buffer size for getmntent_r

Java will only use 1024 byte buffer for parsing mounts. Unlike glibc
will musl return error when this is not big enough instead of truncating
it.

We solve it by allocating a much bigger buffer.

fixes #7093

We also build without precompiled headers, which does not work eith PIE.

History

#1 - 05/25/2017 01:33 PM - Natanael Copa

```
    char buf[1024];
    int buflen = sizeof(buf);
```

The buffer allocated is only 1024 bytes, and the data needed for getmntent_r is bigger than that. Looks like glibc will silently truncate the data. Glibc will silently give you corrupted data.

#2 - 05/25/2017 01:40 PM - Natanael Copa

I think we may be able to fix this in openjdk. Do you have a java testcase that triggers the issue?

#3 - 05/26/2017 09:31 AM - Dimitrios Liappis

> I think we may be able to fix this in openjdk. Do you have a java testcase that triggers the issue?

Sure. You can use:

```
cat >MyMain.java <<EOF
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class MyMain {
    public static void main(String[] args) throws IOException {
        System.out.println(Files.getFileStore(Paths.get(args[0])));
```

```
    }
}
EOF
```

followed by:

```
javac MyMain.java && java MyMain /
```

Note that you need the openjdk8 apk package installed as well as `/usr/lib/jvm/default-jvm/bin` in the PATH.

This, for example, fails on docker-ce with the overlay2 storage driver but works with the device-mapper storage driver.

#4 - 05/26/2017 09:35 AM - Dimitrios Liappis

> I think we may be able to fix this in openjdk. Do you have a java testcase that triggers the issue?

Sure. You can use:

```
cat >MyMain.java <<EOF
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class MyMain {
    public static void main(String[] args) throws IOException {
        System.out.println(Files.getFileStore(Paths.get(args[0])));
    }
}
EOF
```

followed by:

```
javac MyMain.java && java MyMain /
```

Note that you need the openjdk8 apk package installed as well as `/usr/lib/jvm/default-jvm/bin` in the PATH.

This, for example, fails on docker-ce with the overlay2 storage driver but works with the device-mapper storage driver with the alpine images (and works in all cases with glibc images).

#5 - 05/30/2017 02:40 PM - Natanael Copa

- Target version set to 3.6.1

#6 - 05/30/2017 03:07 PM - Przemysław Pawełczyk

ncopa mistyped issue number in its 874cfb1f, so it didn't show in this ticket automatically.

#7 - 05/30/2017 03:54 PM - Natanael Copa

There is a problem in centos too actually. Here is how to re-produce:

```
dd if=/dev/zero of=a.img bs=1M count=200
mkfs.ext4 a.img
p=$(for i in $(seq 0 265); do printf "%.3d/" $i; done)
mkdir -p $p
mount -o loop a.img $p
touch $p/test.txt
java MyMain $(find -name test.txt)
```

#8 - 05/30/2017 03:58 PM - Natanael Copa

fix is in openjdk8 8.131.11-r1 in the mentioned commit 874cfb1f.

Test with docker:

```
/ # java MyMain /
/ (overlay)
/ # grep overlay /proc/mounts  | wc -c
1816
```

#9 - 05/30/2017 04:17 PM - Natanael Copa

- Status changed from New to Resolved

- % Done changed from 0 to 100

Applied in changeset 189a271214ef539691db0b2647a7246e6a8b8343.

#10 - 06/01/2017 07:20 PM - Natanael Copa

- Status changed from Resolved to Closed