ECE590 Enterprise Storage Architecture Lab #4: Security, Workloads, and Data Forensics

Directions:

- This assignment will be completed in your groups. However, **every member of the group must be fully aware of every part of the assignment**. Further, while you can discuss concepts with other groups, actual steps and answers should not be shared between groups.
- The assignment will ask for short answers or screenshots; this material should be collected in a file called `ece590-group<NUM>-lab4.pdf`, where <NUM> is your group number, and submitted via Sakai. *Word documents will not be accepted*. Anything you need to include in this document is highlighted in cyan.

1 At-rest encryption [20pts]

Configure your hard drives as a software RAID5. Put a filesystem on top and write a distinctive text file to the filesystem. Unmount the filesystem. Using a hex dump tool (`hd` or `hexdump`) or `strings`, show that your text content is visible in the raw block device (i.e., by reading `/dev/md0`). Further, show that the file (or its constituent parts) is visible on the underlying real block devices (`/dev/sdb`, `/dev/sdc`, etc.).

Remount the filesystem, delete the file, and unmount the filesystem, showing the commands used. Confirm that the file content is *still* visible in the md0 device and underlying RAID device(s).

Blow away the filesystem by apply disk-level encryption (e.g. with LUKS) to the software RAID device, then make a new filesystem on top of the encrypted block device. Some research will be needed here. Show your process.

Again write a distinctive text file to the filesystem. Using a hex dump tool (`hd` or `hexdump`) or `strings`, show that your text content is ***not*** visible in the raw block device (i.e., by reading `/dev/md0`). Further, show that the file (or its constituent parts) is ***not*** visible on the underlying real block devices (`/dev/sdb`, `/dev/sdc`, etc.).

Note: Tear down the encryption layer before proceeding to Question 3.

2 Workload profiling [20pts]

There is a Linux binary called "`mystery_app`" linked from the course page. Use `strace` to analyze it. What is the size of IO operation it typically does? After the initialization phase, does it do primarily random or sequential IO? Describe how you figured it out.

3 Application benchmarking [30pts]

3.1 Basic setup

Configure your server's hard drives into a software RAID5 and format it with an ext4 filesystem; mount the filesystem. *Note: If the disk encryption you set up in Question 1 is still active, tear it down and use an unencrypted RAID device.*

Install the PostgreSQL database package (directions – you can skip the detailed user setup stuff). Show the relevant commands.

Note: you'll need to switch to the postgres user for the database-focused steps that follow.

Create a database whose backing store is on your RAID5 filesystem (see here for info on creating a database in a particular location). Show the relevant commands.

Research pgbench, a PostgreSQL benchmarking tool that comes with the database. Initialize your newly created database with a scale factor of 100. Show the relevant commands.

Open two command prompts. In one, run "`iostat 1`", which will report IOPS information every second for all block devices. In the other, run a basic 1-client, 10-second-long pgbench test against your database. Verify that the IOPS generated are hitting your RAID5 device and not the OS drive. Show the relevant commands and output from pgbench and iostat.

3.2 Benchmarking

Using `pgbench`, run a series of tests with a runtime 30 seconds, varying the client count in the set {1, 10, 20, 30, ...}. Before each test, clear OS caches with the command like the following:

```
  sudo bash -c "sync; echo 3 > /proc/sys/vm/drop_caches"
```

At low numbers of clients, increasing clients should cause database transactions per second (TPS) to go up, as the database has more to do in parallel. At some point, you will saturate the ability of the server to respond and the TPS will stop increasing. Keep testing until a few datapoints *past* where this happens. Then go back and take two more repetitions of this data, recording it all. Automation is recommended for these tests. Briefly show the process you used to do this testing.

3.3 Analysis

In the end, produce a scatter plot showing clients on the X axis and average TPS on the Y axis. Include error bars showing standard deviation of TPS for each datapoint. Produce a similar graph for latency.

Roughly what is the maximum TPS you were able to get out of the server? What client count corresponds to this maximum?

4 Data forensics [30pts]

A series of disk images is provided on the course site. Download them to a Linux system on which you have root access (your server or a VM will work).

Each image contains a filesystem in which one or more secrets have been written which you must recover.

4.1 Identifying what you have [5pts]

The UNIX utility called '`file`' examines files for common headers and contained structures to identify the most likely filetype. This tool can identify mundane files, such as ZIPs and JPEGs, but also can identify file system images.

Use the '`file`' utility to identify the type of each disk image.

4.2 Image 1 [5pts]

This recovery task is easy: the secret file hasn't been removed at all! Therefore, we can just mount the filesystem and read the data.

Normally, the mount utility can only attach filesystems which reside on block devices, such as disks. However, Linux has a facility called *loop devices* to allow regular files to act as virtual block devices. Research loop devices and mount the first image. Locate the secret text file, and note the image 1's secret in your writeup, showing the steps you used as well.

4.3 Image 2 [5pts]

In this case, the secret file has been deleted. Mount the image and confirm this.

Now, while we could analyze disk structure to try to identify where the data is in the image, we can cheat instead. The content we want is plain ASCII text, so one simple approach we can do is to just dump all the human-readable text inside of the binary image, as most of it will be nulls and non-ASCII-looking binary junk. A common UNIX utility exists to just that: `strings`. Use this tool to also recover the secret content, showing the result in your write-up.

4.4 Image 3 [5pts]

This image also contains deleted secrets, but they're not text. Instead, they're three binary files of different commonly used formats.

As we learned in class, deleting files is just a minor metadata operation, and while this binary content won't show up easily in `strings`, it's likely still present, even if the metadata doesn't point to it.

You could open the image in a hex editor, look up the file system spec, and start to reverse engineer where the data is. However, instead, we can note that the data we want is likely in a common, recognizable format, and therefore just scan the image for regions of data that match well-known formats. There are many tools for this purpose, including the Linux tools '`foremost`', '`photorec`', and '`testdisk`'.

Using one of these tools (or another of your choosing), retrieve at least two of the three secrets from this image, noting in your write-up how you did it.

NOTE: For some reason, these automated tools had trouble identifying the third secret I embedded. I'll give +10 extra credit points if you can get all three secrets.

4.5 Image 4 [5pts]

Okay, at this point, whoever is trying to delete this data is getting upset that we keep being able to retrieve it. They've moved to ext4 for reliability and have made use of the UNIX `shred` command to destroy the secret file.

Look up the `shred` documentation. What does it do? The authors warn that there are some scenarios in which this command will not actually destroy the real data; what are they?

Using any combination of the techniques above, recover the secret in this image, noting the steps you used.

4.6 Image 5 [5pts]

The person making these images has made a small configuration change to how they use their filesystem. Now when they use the `shred` command, the data really is gone. Try the technique you used for Image 4 – you'll find it no longer works.

What simple filesystem configuration change did they likely make?

NOTE: This image does *not* contain a retrievable secret!