SSSS Cozart demo handout

There are four major steps in this tutorial.

1. Setting up the environemnt
2. Trace the workload to generate configlets
3. Compose the configlets
4. Test the debloated kernel

Setup

If you have the container running already (verified by `sudo docker ps -a | grep cozart`), you can use the following commands to enter the container.

```
sudo docker start cozart
sudo docker exec -it cozart /bin/bash
```

Starting from this point, you are in the container and should see a bash prompt.

Execute the following commands in the container.

```
cd /Cozart
source constant.sh
make build-db # parse the linux source to extract the relationships between the
configuration options and code
```

In this demo, we will be debloating a kernel for Nginx so the workload looks like the following.

```
...
service nginx start
sleep 3;
for i in `seq $itr`; do
    ab -n $reqcnt -c 100 127.0.0.1:80/index.html;
done
service nginx stop
...
```

- `make build-db` parses the Linux source files to extract the relationships between source files and kernel configurations. It generates two files: `directives.db` and `filename.db`.
- `directives.db` contains the mapping of C directives and source code lines as the following:

```
linux-cosmic/arch/x86/boot/compressed/head_32.S:264:#ifdef CONFIG_EFI_STUB
linux-cosmic/arch/x86/boot/compressed/head_32.S:271:#endif
linux-cosmic/arch/x86/boot/compressed/head_64.S:101:#ifdef CONFIG_RELOCATABLE
linux-cosmic/arch/x86/boot/compressed/head_64.S:110:#endif
```

- `filename.db` contains the mapping of Makefiles and configuration as the following:

```
linux-cosmic/drivers/iio/pressure/zpa2326.c CONFIG_ZPA2326
linux-cosmic/drivers/zorro/names.c CONFIG_ZORRO_NAMES
linux-cosmic/drivers/net/ethernet/8390/zorro8390.c CONFIG_ZORRO8390
linux-cosmic/drivers/zorro/zorro.c CONFIG_ZORRO
linux-cosmic/drivers/zorro/zorro-sysfs.c CONFIG_ZORR
```

Trace

```
./job.sh trace boot # generate a baselet
./job.sh trace nginx # generate an applet for apache (the executed workload in the VM is in /benchmark-scripts/nginx.sh)
```

We finally prepare all necessary files for debloating including QEMU, kernel source, compiled kernel, application and its workload. We are about to trace our workload to debloat the kernel. Cozart divides the configuration into `baselet` and `applet`.

- A `baselet` is the configuration to boot and a `applet` is the configuration option for the application.
- `./job.sh trace` runs QEMU with trace enabled (emulation mode). QEMU would log all executed PC locations then a script will translate the PC into kernel source code lines by `addr2line` and will map the lines to configuration options.

We generate the baselet by running `./job.sh trace boot`. This will start up a VM and trace it until it successfully boots. A baselet, `config-db/linux-cosmic/cosmic/boot.config`, will be generated.

We generate the applet for Nginx by running `./job.sh trace nginx`. This will start up a VM and trace it until the designated Nginx workload is finished. An applet, `config-db/linux-cosmic/cosmic/nginx.config`, will be generated.

The way to distinguish the phase of booting and the phase of running Nginx workload is a marker program. This program makes the program counter to a magic location i.e., `0x333333333000` and `0x222222222000`. We need to execute the marker before and after the workload to know the phase of workload by inspecting the trace.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>

int main(int argc, char **argv) {
    char *ptr = NULL;
    if(argc == 1) {
            ptr = mmap(0x333333333000, 0x1000, PROT_READ|PROT_WRITE|PROT_EXEC,
                            MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    }
    else {
            ptr = mmap(0x222222222000, 0x1000, PROT_READ|PROT_WRITE|PROT_EXEC,
                            MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    }

    memset(ptr, 0xc3, 0x1000);
    ((void(*)())ptr)();
    printf("ptr: %p\n", ptr);
}
```

Compose

```
./job.sh compose nginx # compose apache applet with boot baselet
```

Now, we have a baselet and an applet for Nginx. We need to compose them together for the final kernel configuration by running `./job.sh compose ngnix`.
A debloated kernel will be built and placed in `kernelbuild/linux-cosmic/cosmic/nginx`. (You will notice the building time is a lot shorter compared to building the vanilla kernel.).

Test

You can use the following command to test if the debloated kernel works.

```
$qemubin -smp $cores -m $mem -cpu $cpu \
    -drive file="$workdir/qemu-disk.ext4,if=ide,format=raw" \
    -kernel $kernelbuild/$linux/$base/nginx/vmlinuz* -nographic \
    -no-reboot -append "nokaslr panic=-1 console=ttyS0 root=/dev/sda rw init=/benchmark-scripts/nginx.sh"
```

We compare the kernel size by running the command. `size $linux/vmlinux kernelbuild/linux-cosmic/cosmic/base/vmlinux`
The first line represents the Nginx kernel that we just built and the second line is the vanilla kernel. We see a 24% reduction in size.

```
   text    data     bss     dec     hex filename
13576072        5792766 1187840 20556678        139ab86 linux-cosmic/vmlinux
15825782        8519002 2576384 26921168        19ac8d0 kernelbuild/linux-cosmic/cosmic/base/vmlinux
```

Starting from scratch (Not necessary for this session because we prepare the AWS image)

The environment is packed in a container image. If starting from scratch, you can use the following commands to setup the environment. These commands clone the git repository, switch to `s4_demo` branch, build the container image from a Dockerfile and, finally, run a container.

```
git clone https://github.com/hckuo/Cozart.git ~/Cozart
cd ~/Cozart/docker
git checkout s4_demo
sudo docker build -f Dockerfile -t cozart-env:latest .
cd ~/Cozart
sudo docker run -v $PWD:/Cozart --privileged -it --name cozart cozart-env /bin/bash
```

In the docker container:

```bash
cd /Cozart
source constant.sh
make $mnt; make $disk # set-up mnt folder and qemu disk
make debootstrap # create a rootfs for the VM
make setup-qemu # patch the qemu to enable PC tracing
make setup-linux # clone the linux source
make build-base # build the vanilla kernel as the baseline
make build-db
```

The above lists all commands necessary to setup the environment.

- `constant.sh` contains constant variables such as the kernel and disk path.
- We then need to patch QEMU in order to enable tracing by `make setup-qemu`.
- We download the Linux source files and build our vanilla kernel by `make setup-linux`.
- `make debootstrap` utilizes the system command, `deboostrap` to create disk image for tracing and it prepares necessary files such as the application and its workload.
- `make build-base` builds a vanilla kernel by Ubuntu cosmic config and store it at `kernelbuild/`. (This can take a while.) This vanilla kernel will be used as the baseline for debloating.
- `make build-db` parses the Linux source files to extract the relationships between source files and kernel configurations. It generates two files: `directives.db` and `filename.db`.