

BPF dev hacks

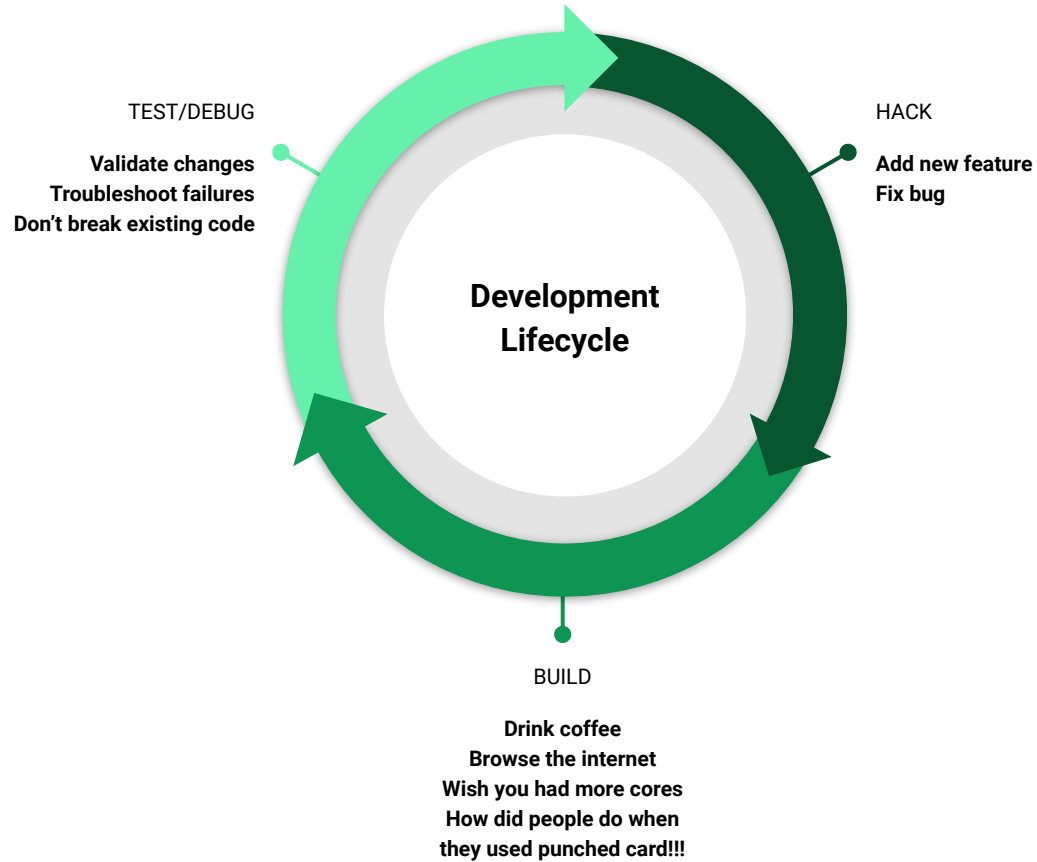
Overview

- Local development
- Getting a shell and repro environment using BPF CI artifacts
- Troubleshooting cross-architecture problem

Tooling

- Recent Ubuntu... but not too recent... < 24.04
 - Make most bits easy to install.
 - Easy cross-compilation support
- [danobi/vmtest](#) >= v0.12.0
 - Used by BPF CI.
 - Easy one-liner mode.
 - Useful to put in `git bisect run` script.
 - Requires `qemu-guest-agent` in guest FS.
- docker or [docker2rootfs](#)
 - create rootfs from docker images.
- `binfmt_misc`
 - Run cross-platform binaries seamlessly
- `qemu-system-{x86,arm,s390x} qemu-user-static`

Local development



Local development - Build kernel and BPF selftests

```
# Setup BPF selftest Kconfigs
cat tools/testing/selftests/bpf/config{,.vm,.$(uname -m)} > .config
make olddefconfig
# Build the kernel
make -j$((4* $(nproc)))
# Build BPF selftests
make -j$((4* $(nproc))) -C tools/testing/selftests/bpf
# Test
pushd tools/testing/selftests/bpf
for t in ./test_verifier ./test_maps ./test_progs
do
    $t
done
```

Local development - run tests

```
# run all test_progs in VM export result to results.json
vmtest -k $(make -s image_name) "./run_bpf_test.sh ./test_progs -J results.json"
```

```
# run a specific test_progs in a VM
vmtest -k $(make -s image_name) "./run_bpf_test.sh ./test_progs -t mytest"
```

```
# test_verifier
vmtest -k $(make -s image_name) "./run_bpf_test.sh ./test_verifier"
```

```
# test_maps
vmtest -k $(make -s image_name) "./run_bpf_test.sh ./test_maps"
```

```
#!/bin/bash
# run_bpf_test.sh

set -ex
grep -qs 'bpffs' /proc/mounts || /bin/mount bpffs /sys/fs/bpf -t bpf
ip link set lo up
cd $(dirname "$0")/tools/testing/selftests/bpf

set +ex

exec "$@"
```

Local development - shell

```
# get a shell in VM, CWD mounted under /mnt/vmtest
vmtest -k $(make -s image_name) -
root@(none):/#

# Finish setting up the VM
/mnt/vmtest/run_bpf_test.sh bash
root@(none):/mnt/vmtest/tools/testing/selftests/bpf#
# troubleshoot directly from within the VM
# what's available in the host is available in the VM
...
...
```

```
# tcpdump -r /tmp/lwt_trace.cap
reading from file /tmp/lwt_trace.cap, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144
Warning: interface names might be incorrect
01:18:15.602769 link_err Out IP6 fe80::acb7:53ff:fea9:8996 > ip6-allrouters: ICMP6, router solicitation, length 16
01:18:19.499438 link_err Out IP6 fe80::acb7:53ff:fea9:8996 > ip6-allrouters: ICMP6, router solicitation, length 16
01:18:27.507122 link_err Out IP6 fe80::acb7:53ff:fea9:8996 > ip6-allrouters: ICMP6, router solicitation, length 16
01:18:42.221055 link_err Out IP6 fe80::acb7:53ff:fea9:8996 > ip6-allrouters: ICMP6, router solicitation, length 16
01:19:12.941362 link_err Out IP6 fe80::acb7:53ff:fea9:8996 > ip6-allrouters: ICMP6, router solicitation, length 16
```

```
# Reproduce lwt_redirect test
ip netns add ns_lwt_redirect
ip netns exec ns_lwt_redirect bash
ip tuntap add mode tap tap0
ip link add link_err type dummy
ip addr add dev lo 10.0.0.1/32
ip link set link_err up
ip link set tap0 up
tcpdump -i any -s 0 -c 20 -w /tmp/lwt_trace.cap &
ip route add 10.0.0.0/24 \
    dev link_err encap bpf xmit \
    obj test_lwt_redirect.bpf.o sec redir_ingress
ip route add 20.0.0.0/24 \
    dev link_err encap bpf xmit \
    obj test_lwt_redirect.bpf.o sec redir_egress
ping -c1 -W1 -s 100 20.0.0.0.$(ip -json a s tap0 | jq ".[] |
.ifindex")
```

Local development - fast test iteration

VM

```
# get a shell in VM, CWD mounted under
/mnt/vmtest
vmtest -k $(make -s image_name) -
root@(none):/#
```

```
# Finish setting up the VM
/mnt/vmtest/run_bpf_test.sh bash
root@(none):/mnt/vmtest/tools/testing/selftests
/bpf#
# Run test
./test_progs -t mytest
FAIL
...
...
...
./test_progs -t mytest
SUCCESS
```

Host

```
# print debug^W^Wfix and recompile tests
make -j$((4* $(nproc))) \
    -C tools/testing/selftests/bpf \
    test_progs
```


Local development - git bisect

```
# Initialize a bisect
git bisect init && git bisect good good_rev && git bisect bad bad_rev
#
git bisect run ./bisect_script.sh
# seat back and relax
```



```
#!/bin/bash
# bisect_script.sh

make -j$((4* $(nproc)))
# skip if we could not build
test $? -ne 0 && exit 125

# same for selftests
make -j$((4* $(nproc))) -C tools/testing/selftests/bpf test_progs
test $? -ne 0 && exit 125

vmtest -k $(make -s image_name) "./run_bpf_test.sh ./test_progs -t mytest"
exit $?
```

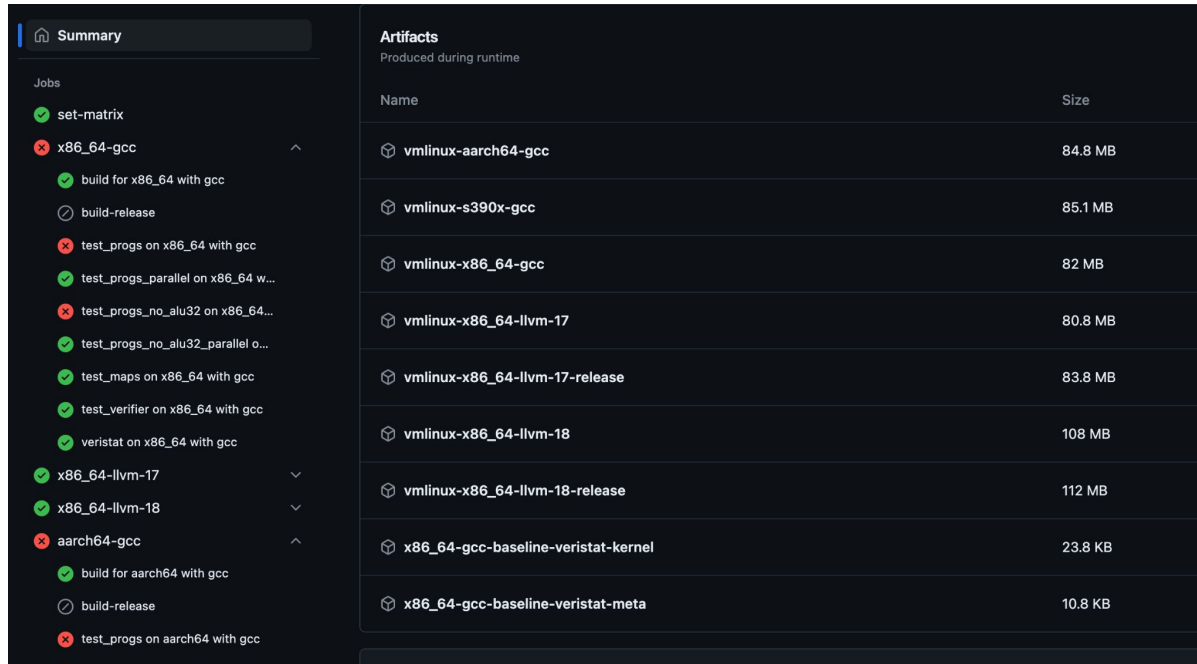
<https://chantra.github.io/bpfcitools/bpf-local-development.html>

BPF CI hacks

- Download artifacts from GitHub
- Pull GH runner's rootfs
- Run test in vmtest
- poke around

BPF CI hacks - download artifacts

<https://github.com/kernel-patches/bpf/actions/runs/9024163551#artifacts>



The screenshot displays the GitHub Actions interface for a workflow run. On the left, the 'Summary' tab is active, showing a list of jobs with their status (success or failure). On the right, the 'Artifacts' tab is active, showing a table of artifacts produced during runtime.

Name	Size
vmlinux-aarch64-gcc	84.8 MB
vmlinux-s390x-gcc	85.1 MB
vmlinux-x86_64-gcc	82 MB
vmlinux-x86_64-llvm-17	80.8 MB
vmlinux-x86_64-llvm-17-release	83.8 MB
vmlinux-x86_64-llvm-18	108 MB
vmlinux-x86_64-llvm-18-release	112 MB
x86_64-gcc-baseline-veristat-kernel	23.8 KB
x86_64-gcc-baseline-veristat-meta	10.8 KB

<https://chantra.github.io/bpfcitools/bpfc-troubleshooting.html>

BPF CI hacks - download artifacts

<https://github.com/kernel-patches/bpf/actions/runs/9024163551#artifacts>

```
gh run -R <repo> download <run_id> -n <artifact_name> -D /download/dir
```

```
gh run -R kernel-patches/bpf download 9024163551 -n vmlinux-s390x-gcc -D /download/dir
```

```
tar -C /download/dir -I zstd -xvf /download/dir/vmlinux-s390x-gcc.tar.zst
```

- Kernel in /download/dir/kbuild-output/arch/*/boot/
- Selftests in /download/dir/selftests/bpf/

BPF CI hacks - download rootfs

```
# docker-less
```

```
docker2rootfs --image kernel-patches/runner -r main-s390x -o /download/dir/main-s390x -a s390x
```

```
# docker
```

```
mkdir /download/dir/main-s390x
```

```
docker pull ghcr.io/kernel-patches/runner:main-s390x --platform s390x
```

```
DOCKER_IMG=$(sudo docker create ghcr.io/kernel-patches/runner:main-s390x)
```

```
docker export "${DOCKER_IMG}" | tar -C /download/dir/main-s390x -xf -
```

```
docker rm "${DOCKER_IMG}"
```

BPF CI hacks - Run tests

```
cd /download/dir
```

```
# Get a shell
```

```
vmtest -k kbuild-output/arch/s390/boot/bzImage -r main-s390x -a s390x -
```

```
# within the VM
```

```
cd /mnt/vmtest/selftests/bpf
```

```
./test_progs -t xdpwall
```

```
...
```

```
# one liner
```

```
vmtest -k kbuild-output/arch/s390/boot/bzImage -r main-s390x -a s390x \  
"cd /mnt/vmtest/selftests/bpf && ./test_progs -t xdpwall"
```

Cross-platform hacks - setup

```
XPLATFORM="s390x"
```

```
# Set up repo for s390x, it is only available from the `ports` repo.
```

```
cat <<EOF >> /etc/apt/sources.list.d/${XPLATFORM}.list
```

```
deb [arch=${XPLATFORM}] http://ports.ubuntu.com/ubuntu-ports $(lsb_release -c -s) main restricted
```

```
deb [arch=${XPLATFORM}] http://ports.ubuntu.com/ubuntu-ports $(lsb_release -c -s)-updates main restricted
```

```
EOF
```

```
# Add the architecture
```

```
dpkg --add-architecture s390x
```

```
apt update
```

```
apt install -y g{cc,++}-${XPLATFORM}-linux-gnu \
```

```
  {libelf-dev,libssl-dev,pkgconf}:${XPLATFORM} \
```

```
  qemu-user-static qemu-system-{x86,arm,s390x}
```

Cross-platform hacks - build

```
XPLATFORM="s390x"
```

```
XARCH="s390"
```

```
cd ~/bpf-next
```

```
export KBUILD_OUTPUT="$(pwd)/kbuild-${XPLATFORM}"
```

```
mkdir "${KBUILD_OUTPUT}"
```

```
# Currently... we need to hack that up a bit to work cross-endianness for generating the right vmlinux.h.
```

```
curl https://gist.githubusercontent.com/chantra/72a12644074444e4edffe2cfd0e3138e/raw > xcompile.patch
```

```
patch -p1 < xcompile.patch
```

```
# Setup the config for the foreign architecture
```

```
cat tools/testing/selftests/bpf/config{,.vm,}.${XPLATFORM} > ${KBUILD_OUTPUT}/.config
```

```
# Build!
```

```
make ARCH="${XARCH}" mrproper
```

```
make ARCH="${XARCH}" CROSS_COMPILE="${XPLATFORM}-linux-gnu-" -j$((4 * $(nproc))) olddefconfig
```

```
make ARCH="${XARCH}" CROSS_COMPILE="${XPLATFORM}-linux-gnu-" -j$((4 * $(nproc))) all
```

```
make ARCH="${XARCH}" CROSS_COMPILE="${XPLATFORM}-linux-gnu-" -j$((4 * $(nproc))) -C tools/testing/selftests/bpf
```

<https://chantra.github.io/bpfcitools/bpf-cross-compile.html>

Cross-platform hacks - generate a rootfs

```
XPLATFORM="s390x"
```

```
# docker2rootfs
```

```
docker2rootfs -R registry-1.docker.io -i ${XPLATFORM}/ubuntu -r ${VERSION_ID} -o s390x_rootfs
```

```
# or docker
```

```
docker pull ${XPLATFORM}/ubuntu:${lsb_release -c -s} --platform ${XPLATFORM}
```

```
DOCKER_IMG=$(sudo docker create ${XPLATFORM}/ubuntu:${VERSION_ID})
```

```
docker export "${DOCKER_IMG}" | tar -C s390x_rootfs -xf -
```

```
docker rm "${DOCKER_IMG}"
```

Cross-platform hacks - add toolchain in rootfs

```
# We need to chroot and install a few useful packages
```

```
cp /etc/resolv.conf s390x_rootfs/etc/
```

```
sudo chroot s390x_rootfs
```

```
# Within chroot
```

```
mount -t devtmpfs -o nosuid,noexec dev /dev
```

```
mount -t tmpfs tmpfs /tmp
```

```
apt update
```

```
DEBIAN_FRONTEND=noninteractive apt-get install -y qemu-guest-agent ethtool keyutils iptables gawk libelf1 zlib1g libssl3 iproute2
```

```
finit-sysv
```

```
umount /tmp /dev
```

```
# exit chroot
```

```
exit
```

Cross-platform hacks - run tests

```
vmtest -k "$(make ARCH="${XARCH}" O="${KBUILD_OUTPUT_DIR}" -s image_name)" -r /tmp/s390x_rootfs/ -a s390x "uname -m"  
# Shell  
vmtest -k kbuild-output/arch/s390/boot/bzImage -r main-s390x -a s390x -  
# within the VM  
cd /mnt/vmtest/selftests/bpf  
./test_progs -t xdpwall  
...  
# one liner  
vmtest -k kbuild-output/arch/s390/boot/bzImage -r main-s390x -a s390x \  
"cd /mnt/vmtest/selftests/bpf && ./test_progs -t xdpwall"
```