Testing Bleeding Edge Kernels

Paul Moore

August 2016

Introduction

- Who am I?
  - Paul Moore
    - Email: paul@paul-moore.com / pmoore@redhat.com
    - Twitter: @securepaul
  - SELinux, audit, labeled networking kernel maintainer
  - Created and maintain the libseccomp project
- What am I talking about?
  - Kernel testing problems and improvements
    - Regular testing of development patches
    - Making it easier for users to test development patches

Kernel Development Cycle

- Developer creates a patchset and posts it upstream
- Patches are reviewed and changes are made
- Patches are merged into a subsystem “next” branch
  - The specifics vary by subsystem
- Linus pulls the “next” branches into his tree
  - Starts immediately after the latest kernel is released
  - The “merge window” lasts for two weeks
- Approximately 8 week RC test cycle before release
  - Weekly kernel RC releases, typically on Sunday

Kernel Testing Today

- Kernel RC releases are packaged for Fedora Rawhide
  - Multiple releases each week to sync with Linus
- Subsystem “next” branches not packaged for Fedora
  - The linux-next repository merges “next” branches daily
  - Primarily focused on exposing merge conflicts
  - Some automated testing exists, coverage unclear
  - General user testing appears to be rare
  - Usually doesn’t catch subsystem interaction problems

Problems with the Current Approach

- Widespread user testing doesn’t happen until RC1
  - Less than 8 weeks to find and solve any problems
  - Serious bugs incur the wrath of LKML and Linus
- The “next” branches based on stable/old code
  - Based on released kernel, ~8 weeks old (that’s old!)
- Subsystems may lack good regression testing
  - Relies heavily on code review and ad hoc testing
  - No way to easily test subsystem interactions

Kernel Testing Aspirations

- Regular testing of the latest upstream developments
  - Test as close to development as possible
- Make test kernels more accessible to users
  - Not all users are comfortable building kernels
- Solution needs to be cheap, easy, and portable
  - Automated build and test processes
  - Close to zero cost, both in terms of time and money
  - Needs to run on a laptop with uncertain connectivity

Continuous Integration as a Solution

- Connects development with test and deployment
  - New commits trigger new rounds of test runs
  - Successful test runs trigger deployment
- Constant stream of new, verified releases
  - Some projects release multiple times a day
- Many services and projects available
  - Very popular with web and application developers

Continuous Integration Problems

- Many CI frameworks require infrastructure
  - Requires network connectivity
  - Hardware and/or management cost could be high
- Does not appear currently well suited to kernel testing
  - Need to reboot multiple times with different kernels
  - Need to gracefully handle kernel panics

Simplified CI for Kernel Developers

- Create/find a set of easily run regression tests
  - Self contained, single system tests
  - Individual tests can be isolated for use as reproducers
- Automate patching of Fedora kernel SRPMs
- Automate building Fedora kernel RPMs
- Automate RPM distribution to a public repository
  - Kernel packages available via yum / dnf
- Automate testing via VMs
  - Run locally via QEMU or via remote hosting service

Progress So Far ...

- Usable test suites for SELinux and audit
  - Updated the selinux-testsuite project
  - Created the audit-testsuite project
- Fedora kernel patching automation
  - Bash scripts generate a patched kernel SRPM
  - Optionally submit scratch and Copr builds
- Copr repository for kernel build and distribution
  - Weekly builds and test runs for the past year

Lessons Learned

- Regular regression testing does work (if you do it!)
  - Verify new patches and subsystem interactions
  - Less stress during the merge window and RC cycle
- Automated patching works better than expected
  - Merge conflicts are rare and easily resolved
  - Considering further automation in this area
- Copr is a great tool but it has reliability issues
  - Build failures due to infrastructure problems
  - According to Copr team, things will improve

Future Improvements

- Automate everything (or as much as I can)
  - VM test execution is still in the early stages
  - Automatic build and test triggers
    - Kernel changes trigger a new build
    - New build triggers a new test run
- Increase test coverage (within reason)
  - New functionality must provide matching tests
  - Better support for testing existing functionality
- Additional security subsystems beyond SELinux/audit

Project Links

- The selinux-testsuite project
  - https://github.com/SELinuxProject/selinux-testsuite
- The audit-testsuite project
  - https://github.com/linux-audit/audit-testsuite
- The pcmoore/kernel-secnext Copr kernels
  - https://copr.fedorainfracloud.org/coprs/pcmoore/kernel-secnext
- Kernel patching automation scripts
  - https://github.com/pcmoore/copr-pkg_scripts

Installing the Fedora Kernel Patch Scripts (#1)

- Install and configure the Copr client
  - Install the client via dnf

    ```
    dnf install copr-cli
    ```

  - Get an access token, requires Fedora account
    - https://copr.fedorainfracloud.org/api
- Clone the GitHub repository into its own directory

  ```
  git clone https://github.com/pcmoore/copr-pkg_scripts.git
  ```

Installing the Fedora Kernel Patch Scripts (#2)

- Create and configure a new Copr project directory

```
mkdir copr-project
cd copr-project
ln -s ../copr-pkg_scripts/pcopr_patch .
ln -s ../copr-pkg_scripts/pcopr_srpm-kernel .
cp ../copr-pkg_scripts/pcopr.config .
$EDITOR pcopr.config
```

- Clone the upstream project and add any remotes

```
cd copr-project
git clone <project URL>
```

- Clone the Fedora package repository

```
cd copr-project
fedpkg clone <package>
```

Running the Fedora Kernel Patch Scripts

- Generate the patches from upstream

```
cd copr-project
./pcopr_patch
```

- Create a patched SRPM and submit to Copr

```
cd copr-project
./pcopr_srpm-kernel -b
```