Core Infrastructure Initiative Badge Program

Paul Moore

January 2017

Introduction

- Who am I?
  - Twitter: @securepaul
  - Email: paul@paul-moore.com / pmoore@redhat.com
  - SELinux, audit, labeled networking kernel maintainer
  - Created and maintain the libseccomp project
- What am I talking about?
  - Core Infrastructure Initiative's Badge Program
  - Improving the quality / security of Open Source projects

Core Infrastructure Initiative (CII)

- Linux Foundation sponsored project
  - Supports critical Open Source projects
  - Created in response to Heartbleed (2014)
  - https://www.coreinfrastructure.org
- CII efforts focused on improving quality / security
  - Evaluate Open Source projects for risk
  - Help fund developers and projects
  - Supports secure development and testing tools
  - Provides developer education

CII Best Practices Badge Program

- Help educate and evaluate project quality / security
  - Secure development practices
  - Open collaboration mechanisms
  - Proper documentation
- Voluntary program
  - Projects self-certify via the CII badge website
    - https://bestpractices.coreinfrastructure.org
- Anyone can contribute comments and ideas
  - Open Source project hosted on GitHub

CII Best Practices Case Study - libseccomp

- Modern Linux Kernels support flexible syscall filtering
  - Commonly called “seccomp” or “seccomp-bpf”
  - Kernel enforces application provided BPF filter code
  - Supported on most architectures/ABIs
- libseccomp designed to make filtering easier
  - Simple API instead of seccomp BPF filter code
  - Abstracts away architecture/ABI specifics
  - Generates optimized filter code
- See presentation from DevConf.cz 2014

CII Best Practices Focus Areas

- Open Source
- Change control
- Reporting
- Quality
- Security
- Analysis

Open Source

- License
  - FLOSS required, OSI suggested
- Public website
  - Instructions on how to install/use and contribute
  - Support for HTTPS using TLS
- Project documentation
  - Interface/API must be documented
  - All materials must be written in English
- At least one public discussion mechanism
  - Mail list, forum, etc.

Open Source - libseccomp

- **License [PASS]**
  - LGPL v2.1
- **Public website [PASS]**
  - README displayed on front page (GitHub)
  - Accessible via TLS v1.2 (GitHub)
- **Project documentation [PASS]**
  - Interface/API fully documented via man-pages
  - All materials written in English
- **At lease one public discussion mechanism [PASS]**
  - Mail list (Google Groups), issue tracker (GitHub)

Change Control

- Public source repository
  - Track authors and dates of individual changes
  - Distributed version control mechanism preferred
- Versioning
  - Releases must have unique version numbers
  - Semantic versioning preferred
- Release notes
  - Human readable summary
  - Identify all vulnerabilities that have been fixed

Change Control - libseccomp

- **Public source repository [PASS]**
  - All development occurs in the git repository (GitHub)
- **Versioning [PASS]**
  - Tagged releases follow Semantic Versioning
  - Branches for each MAJOR/MINOR release stream
- **Release notes [PASS]**
  - Summary of changes in CHANGELOG
  - Release notes for each tagged release (GitHub)

Reporting

- Bug reporting
  - Provide mechanism for reporting bugs
  - Public archive of bug reports and responses
  - Majority of reports acknowledged
- Vulnerability reporting
  - Documented process for reporting
  - Maximum response time less than 14 days

Reporting - libseccomp

- **Bug reporting [PASS]**
  - Bug reporting via mail list and issue tracker
  - Mail list and issue tracker are publicly archived
  - All reports are acknowledged in a timely manner
- **Vulnerability reporting [FAIL]**
  - No dedicated guidance on reporting vulnerabilities
  - SOLUTION: additional documentation

Quality

- Build system
  - Working build system using common FLOSS tools
- Build warnings
  - Use compiler warnings, “safe” mode, “lint” tools
  - Warnings must be addressed
- Automated test suite
  - Tests released under FLOSS license
  - Tests should provide full code coverage
  - Continuous integration methods preferred

Quality - libseccomp

- **Build system [PASS]**
  - Build system uses autotools, make, gcc, cython
- **Build warnings [PASS]**
  - Build uses ‘-Wall’ by default, all warnings resolved
- **Automated test suite [MIXED]**
  - Test suite included in the main repository
  - Unknown code coverage
    - SOLUTION: add coverage analysis to test suite (gcov?)
  - No continuous integration testing
    - SOLUTION: leverage GitHub/CI integration (Travis CI?)

Security

- Secure development
  - Developers educated in secure software design
- Cryptography
  - Use established/reviewed crypto protocols
  - Implementations must be FLOSS licensed
  - Requirements on key lengths and secret storage
- Defend against Man-In-The-Middle (MITM)
  - Project material available via HTTPS, SSH, etc.

Security - libseccomp

- **Secure development [PASS]**
  - Maintainer trained in secure development practices
- Cryptography [N/A]
  - Project does not use any cryptography
- **Defend against Man-In-The-Middle (MITM) [PASS]**
  - All material accessible via HTTPS or SSH (GitHub)

Analysis

- Static analysis
  - Major releases tested and problems resolved
- Dynamic analysis
  - Major releases tested and problems resolved

Analysis - libseccomp

- **Static analysis [PASS]**
  - Each release tested with Coverity
    - Free analysis for Open Source projects
    - All problems resolved prior to release
- **Dynamic analysis [PASS]**
  - Automated test suite includes valgrind tests
    - Runs on all platforms that support valgrind
    - Code changes must pass the entire test suite

More Information

- CII Best Practices Badge Program
  - https://bestpractices.coreinfrastructure.org
- The libseccomp Project
  - https://github.com/seccomp

- Me
  - Twitter: @securepaul
  - Email: paul@paul-moore.com
  - Web: http://www.paul-moore.com