Syscall Filtering and You

Paul Moore

February 2014

What is Linux Syscall Filtering

- Triggers the kernel to take action on syscalls
  - Kill the calling process
  - Signal the calling process
  - Signal the parent tracing process
  - Cause the syscall to return a specific “errno” value
- Configured on a per-process basis
  - Configured at application run-time
  - Inherited by the process' children
  - Enforced by the kernel

Why You Should Care

- Bugs are both inevitable and a security risk
  1. Application bugs can lead to malicious code execution
  2. Malicious code can exercise kernel vulnerabilities
  3. Kernel vulnerabilities can lead to … @!$&%#!
- Focus on containing bugs and limiting the risks
  - System hardening
  - Access controls
  - … and now, **syscall filtering**

The Start of Linux Syscall Filtering

- Initial syscall filtering functionality, “seccomp mode 1”
  - First appeared in Linux 2.6.23
  - Filtering is enabled on a per-application basis
  - Fixed list of allowed syscalls:
    - read(2), write(2), exit(2), sigreturn(2)
- Not very flexible, not a general purpose solution
  - Unable to filter on architecture/ABI
  - Unable to change the syscall whitelist
  - Unable to filter on syscall arguments

Linux Syscall Filtering Today

- Enhanced syscall filtering, “seccomp mode 2”
  - First appeared in Linux 3.5 for x86, x86_64, and x32
  - Added 32-bit ARM support in Linux 3.8
  - Additional architectures are works in progress
    - 32-bit and 64-bit MIPS (3.15?)
    - 64-bit ARM (patches are available)
- Syscall filters are now defined by a filtering language
  - Applications create their own syscall filter
  - Filter on arch/ABI and any syscall, including arguments
  - Filtering is still done by the kernel

Seccomp Mode 2 and the Filter Language

- Filters are written in BPF (Berkley Packet Filter)
  - Simple low level language, e.g. assembly
  - Originally used in the Linux network socket filter
- Multiple filters can be loaded for each process
  - Syscalls are filtered through the entire filter stack
    - Most restrictive result “wins”
  - Not possible to unload or overwrite a loaded filter
- Syscall filters are inherited across fork(2)/exec(2)
  - Not possible to escape from a syscall filter once set

Language Based Filtering and Complexity

- Drawbacks of language based syscall filtering
  - Syscall tables are different for each architecture
    - Filters are arch/ABI dependent
  - BPF language is 32-bit
    - 64-bit support is possible, but BPF code is more complex
  - Good, non-trivial filters can be cumbersome
    - Similar to coding in assembly language
- Learning curve may be too steep for developers
  - Developers want to focus on their application, not syscall filtering

Making Life Easier with libseccomp

- Provide a programmatic API for seccomp filters
  - Architecture/ABI independent
  - Transparent support for 64-bit architectures
  - No BPF knowledge required
- Provide documentation and examples
  - Currently 22 manpages, 25 tests/examples
- Generate optimized filters without any extra effort
  - Library optimizes filters for size/speed automatically
  - API allows developers to provide syscall priority hints

libseccomp Filter Example

```
int myapp_libseccomp_start(void)
{
        int rc;
        scmp_filter_ctx ctx;

        ctx = seccomp_init(SCMP_ACT_KILL);
        if (ctx == NULL)
                return -ENOMEM;

        rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(open), 0);
        if (rc < 0)
                goto out;
        rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(close), 0);
        if (rc < 0)
                goto out;
        rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 3,
                                                   SCMP_A0(SCMP_CMP_EQ, STDIN_FILENO),
                                                   SCMP_A1(SCMP_CMP_NE, 0x0),
                                                   SCMP_A2(SCMP_CMP_LT, SSIZE_MAX));
        if (rc < 0)
                goto out;

        rc = seccomp_load(ctx);

out:
        seccomp_release(ctx);
        return rc;
}
```


Raw BPF Syscall Filter Example

```
int myapp_seccomp_raw_start(void)
{
        struct sock_filter filter[] = {
                BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 4),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, AUDIT_ARCH_X86_64, 0x00, 0x12),
                BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 0),
                BPF_STMT(BPF_JMP+BPF_JGE+BPF_K, 0x40000000, 0x10, 0x00),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, __NR_open, 0x0e, 0x00),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, __NR_close, 0x0d, 0x00),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, __NR_read, 0x00, 0x0d),
                BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 20),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, 0, 0x00, 0x0b),
                BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 16),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, 0, 0x00, 0x09),
                BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 28),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, 0, 0x00, 0x02),
                BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 24),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, 0, 0x05, 0x00),
                BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 36),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, (SSIZE_MAX >> 32), 0x00, 0x02),
                BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 32),
                BPF_STMT(BPF_JMP+BPF_JEQ+BPF_K, (SSIZE_MAX & 0xffffffff), 0x01, 0x00),
                BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_ALLOW),
                BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_KILL),
        };
        struct sock_fprog prog = {
                .len = (unsigned short)(sizeof(filter)/sizeof(filter[0])),
                .filter = filter,
        };
        if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0) < 0)
                return -errno;
        if (prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, &prog) < 0)
                return -errno;
        return 0;
}
```


How You Can Help (part 1)

- Add syscall filtering to your applications
  - Does your application work with dangerous data?
    - Network applications, virtualization hypervisors, etc.
  - Does your application offer sandboxing / containment?
    - Syscall filtering can provide a new layer of hardening
  - Others?
- Help with libseccomp development
  - Additional automated tests
  - Support for new architecture/ABIs
  - Better tools for developers adding libseccomp support

How You Can Help (part 2)

- Testing
  - Test libseccomp in your environment and applications
  - Test libseccomp on new architectures
  - Try to break libseccomp (please email me the pieces)
- Documentation
  - Improve our existing manpages
  - Develop tutorials and developer guides
  - Help spread the word about libseccomp

More Information

- The libseccomp project
  - Website
    - http://libseccomp.sf.net
  - Mailing list
    - libseccomp-discuss@lists.sourceforge.net
  - Source repository
    - git://git.code.sf.net/p/libseccomp/libseccomp
- Presentation feedback
  - http://devconf.cz/f/108