Locking Down Science Gateways

Steven R Brandt*, Patrick Diehl‡*†
*LSU Center for Computation & Technology, Louisiana State University, Baton Rouge, LA, 70803 U.S.A.
† Department of Physics and Astronomy, Louisiana State University, Baton Rouge, LA, 70803 U.S.A.
‡ Applied Computer Science (CCS-7), Los Alamos National Laboratory, Los Alamos, NM 87545 U.S.A.

***Abstract*—The most recent Linux kernels have a new feature for securing applications: Landlock. Like Seccomp before it, Landlock makes it possible for a running process to give up access to resources. For applications running as Science Gateways, we want to have network access while starting up MPI, but we want to take away network access prior to the reading of parameter files in order to prevent malicious exploits of the gateway code. We explore the usefulness of this tool by modifying and locking down two mature scientific codes: The Einstein Toolkit, and Octo-Tiger.**

***Index Terms*—Science Gateways, security, landlock**

I. Introduction

Science Gateways typically provide a graphical or web interface to scientific code, allowing users who are less savvy about the command line, supercomputers, Slurm, etc. to have ready access to advanced codes. Often, in the interest of democratization, the vetting process for users of the Gateway is less rigorous than a typical user account. Because scientific codes are typically written in C, C++, or Fortran without a thought about security, these applications represent a potential security hazard via buffer overruns, poor input sanitization, etc. Full audits of these codes represent a cost few are willing to undertake.

An ideal solution for these systems would be to sandbox the code, limiting what it can do even if a hacker were to gain control of the running process. Because these applications typically run in a distributed fashion over MPI, they need the ability to turn on the sandbox (and take away the ability to make new connections) after calling `MPI_Init()`. In addition, of course, the sandbox should limit what directories the process can read or write.

The most recent Linux kernel (version 6.9), fortunately, offers a way to lock down an application, *i.e.* a way for an application to give up its access to the network and to files. Can this tool (landlock) readily address the needs of the Science Gateway and make sophisticated scientific applications secure? Can it accomplish this without overburdening the Gateway developer with the need to understand deep things about security or make extensive modifications of their code?

In this paper, we explore the difficulty in securing two codes: The Einstein Toolkit, which we will use to simulate a spherically symmetric neutron star, and Octo-Tiger, which we will use to simulate a white dwarf. These codes have little in common except that they are large C++ codes that explore astrophysical scenarios. We will show that it is relatively easy to modify these codes to employ Landlock.

The paper is structured as follows: Section II discusses security tools and methodologies. Section III briefly introduces the studied scientific applications. Section IV addresses the implementation and testing. Section V shows examples to lock the file systems access and network. Section VI shows some run time measurements for Octo-Tiger with and without `Landlock`. Finally, Section VII concludes the work.

II. Security Tools and Methodologies

Landlock is far from the first tool designed to lock down an otherwise insecure application and prevent it from doing malice. The original Linux had `chroot` to serve this purpose. The `chroot` system call changed the effective root of the process calling it. It thereby gave up access to all files below the new root passed to it. Unfortunately, subsequent calls to `chroot` can *undo* the first call, so it is inadequate even for a low-level sandboxing. The `pivot_root` system call, introduced in Linux 2.3.41, provides an irreversible change of the root directory. While it does make it possible to sandbox direct access to the file system from the current process, it does not prevent the local process from opening network connections, creating IPC resources, etc. So it is, at best, a start at building a sandbox.

Another, more comprehensive effort at limiting what applications can do, is provided by the `seccomp` facility and has been available since Linux 3.17. This tool allows system calls to be selectively blocked and filtered. However, `seccomp` does not claim to be able to sandbox an application, but it does provide a way for an application to give up a wide variety of privileges. One limitation of `seccomp` is that one cannot pass pointers to it. This means it cannot be given character arrays, and this means it cannot be used to limit access to specific files or directories. In principle, it could be combined with `pivot_root` to accomplish this end.

`SELinux` and `AppArmor` provide true sandboxing capabilities, but they must be configured by the root user and provide system-wide restrictions. It should, in theory, be possible to configure them into an image and launch the image from MPI. In principle, these tools could be configured for an Apptainer image and launched by a user. Alternatively, rules could be crafted collaboratively between the sysadmins and the Science Gateway developer. While these tools are fully capable of providing the necessary level of restriction, turning them on after calling `MPI_init()` might, however, prove challenging.

OpenBSD has similiar capabilities to `Landlock` through its `pledge` and `unveil` system calls. However, very few

clusters currently use OpenBSD on their clusters. Namespaces also have the capability of limiting what an app can do, but they were designed more for virtualization than for security. `Landlock`, however, allows programmers the flexibility of controlling when restrictions are turned on and requires neither special permission from sysadmins nor virtualization. We believe that, in many cases, this will make it the best choice for locking down a science gateway.

III. Scientific applications

Although theoretically present in kernel version 5.13, the first version in which `Landlock` was capable of stopping network connections seems to be Fedora 40 running kernel version 6.8.1. We feel that this is a crucial capability for the purposes for preventing bad actors from gaining control of or misusing local resources.

We constructed a function call named `landlockme()` [1] which our applications can call. It is based on an example landlock sandboxing code found here [2]. This code uses environment variables to communicate which directories the application is allowed to read, write, and where (if anywhere) it is allowed to make internet connections and on what ports.

Most scientific codes follow a standard workflow: (1) initialize MPI, (2) then read parameter and/or data files, (3) and then finally produce a result. To secure such an application, one inserts a call to `landlockme()` or the equivalent between steps (1) and (2). For this strategy to be effective, the gateway should not give the user any control over command line arguments to the application, only to the contents of the parameter and input files.

The insertion of this call can be performed in one of two ways: (1) editing the source code, or (2) using PMPI to call `MPI_Init()`.

A. The Einstein Toolkit

The Einstein Toolkit (ET) [3] is a hybrid code constructed from C, C++, and Fortran. It's core infrastructure was first created in 1977 and it has been under continuous development since. While the core infrastructure, Cactus, is generic and could be used for any Cauchy problem, the family of science-specific modules in the ET centers on fully relativistic astrophysical simulations, *e.g.* black holes, neutron stars, supernovae, and cosmology. Cactus provides adaptive mesh refinement (AMR) with subcycling in time. Using the Carpet driver, this is in the form of nested and moving boxes rather than a fully general refinement system.

The test problem we are using in this paper is a TOV star [**?**]. This is a simple, spherically symmetric neutron star which we model on a full 3D Cartesian grid. While this is more computational infrastructure than is needed for such a simple simulation, it is a common test problem that is run to verify code correctness and to teach students about neutron stars and the ET code. The TOV star will exercise all important components of the solvers required for more sophisticated problems.

B. Octo-Tiger

Octo-Tiger is an astrophysical code simulating the evolution of non-relativistic star systems using adaptive octrees [4]. Octo-Tiger simulates the following multi-physics: Gravity is solved using a fast-multipole method (FMM) and the hydro equation is solved using a finite volume method with a fully adaptive mesh refinement (AMR) without subcycling in time (subcycling is avoided because of the need to solve elliptic equations). Octo-Tiger is implemented in C++ using the C++ standard library for parallelism and concurrency (HPX) [5].

IV. Implementation and Testing

We note that while `Landlock` works on Fedora 40, we were unable to get the default installed `valgrind` to work. As far as `valgrind` and the emulated CPU it uses work, the system does not have the capability. The GNU debugger project (gdb) worked with `Landlock`.

A. The Einstein Toolkit

We began by testing very basic MPI codes that exchange simple messages of random data using `MPI_Send` and `MPI_Recv` in order to verify whether our method works.

We were able to show that if `landlockme()` was called before `MPI_Init`, then the application did not run. If we called `landlockme()` after `MPI_Init`, then the application ran without difficulty using MPICH. When we attempted the same test using OpenMPI, `Landlock` blocked an attempt to use shared memory. In principle, we could ask OpenMPI not to do this, or we could change the rules to allow shared memory. For simplicity, we tested our scientific codes using MPICH.

The modification to the Einstein Toolkit was straightforward. We were able to identify the function call `CCTKi_InitialiseCactus` and insert the call to `landlockme()` after the call to a method named `CCTKi_InitialiseDataStructures`. With the addition of this single line of code, our TOV star example was able to run and generate data files.

As a double check that the `Landlock` was indeed active, we ran the tests again, running it in directories it was not supposed to be able to access. As expected, `Landlock` prevented it from reading or writing files.

B. Octo-Tiger

For the Octo-Tiger version without networking, we called `landlockme()` as the first thing after entering the main method. As expected, `Landlock` prevented it from reading or writing files. For the version with networking on, we had to make sure that each MPI rank calls `landlockme()`. Here, we called the function in the initialization of each MPI rank. For debugging purposes, we added simple stdout/stderr messages to the `landlockme()` function. So the function is called for each MPI rank. Adding `LandLock` to Octo-Tiger was straightforward and had no major issues. We have to mention that we had to do code changes unrelated to `LandLock` but specific to GCC 14. We are in the process to prepare a pull request for adding `LandLock` as an optional feature to Octo-Tiger.

Listing 1. Example variables to land lock applications.

```
export LL_FS_RO="/bin:/lib/:$USER/"
export LL_FS_RW="$USER/"
export LL_TCP_BIND=""
export LL_TCP_CONNECT=""
```

C. Generic Science Codes

We note that, with `Landlock`, it is theoretically possible to create a service which runs arbitrary MPI codes on behalf of unknown users. One way to accomplish this would be to accept a user code in the form of a shared library (*i.e.* a `.so` file) with some kind of standard method, *e.g.* `runcode()`. The service would call `MPI_Init`, then `landlockme()`, then it would use `dlopen()` and `dlsym()` to access and run the user method. By using `dlopen()` instead of linking the shared object file, we circumvent the potential problem of constructors being called prior to `landlockme()` in C++'s initialization sequence.

V. Configuration

Listing 1 shows some of the configuration options we used in this study. The first option is `LL_FS_RO`, which takes a list of paths separated by colons. These are the files and directories the application is allowed to read. We gave access to the system-wide installed libraries and executables. The second option `LL_FS_RW` provides a list of files and directories the application is allowed to write to. The third option `LL_TCP_BIND` restricts the port binding and the fourth options `LL_TCP_CONNECT` restricts the ports for connections. We refer to the Linux kernel documentation [6] for more options.

VI. Runtime measurements

Using `landlockme()` should not introduce any overhead to the process. We executed the rotating star problem from Octo-Tiger's test suit to investigate the claim. We adaptive refined the intial mesh four times and executed the simulation for ten steps. Both runs with and without landlock took around 92 seconds. We compiled Octo-Tiger using Spack [7] and restricted it to access the Spack installation directory and the user's home directory to read and write the output files. We could not observe introduced overheads for the Octo-Tiger version with networking.

VII. Conclusion

In this work we have studied the use of `Landlock` for securing scientific applications for use in Science Gateways. Because Science Gateways typically involve taking large, mature, C/C++ and Fortran codes and making them semi-publicaly available on the web, they represent a potential security hazard. These codes usually are based on MPI and follow a pattern of starting up, reading initial data files, computing, then generating results. Modifying such codes to invoke `Landlock` after MPI startup but before the reading of parameter files should secure the code against attackers launching attacks based on input files (e.g. exploiting buffer overruns or unsanitized inputs).

We note, however, that `Landlock` provides no protection against denial of service attacks, *e.g.* using up file space, inodes, file descriptors, etc. While these types of threats can still cause significant problems, they are of a different class. They should not allow user data or password information to be stolen, back doors to be installed, etc.

We have demonstrated that, even with large, complex codes such as Octo-Tiger and the Einstein Toolkit, sandboxing a code with `Landlock` is a relatively straightforward task. Not only was this done with relative ease, it introduced no performance penalties or observable runtime overheads.

Appendix A
Supplementary Materials

Octo-Tiger is available on GitHub [8] and can be compiled with Spack [9]. The scripts and input data to reproduce the runs in Section VI are available on Zenodo [10].

The Einstein Toolkit is free under the GPLv3 license and is available for public download using instructions found at the Einstein Toolkit website [11].

Acknowledgment

The authors would like to thank the IT support staff of the Center for Computation and Technology who setup our test machines for us. Also, we wish to acknowledge the support of NSF grant OAC 2004157 to support work on the Einstein Toolkit. This work was supported by the U.S. Department of Energy through the Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001). LA-UR-24-27511

References

[1] "landlock(7) — linux manual page," last accessed 07/20/2024. [Online]. Available: https://man7.org/linux/man-pages/man7/landlock.7.html

[2] S. R. Brandt, "Sandbox for landlock," last accessed 07/18/2024. [Online]. Available: https://gist.github.com/stevenrbrandt/ced0bd99a90628453cbd899480d435d2

[3] "The einstein toolkit," 2023. [Online]. Available: https://doi.org/10.5281/zenodo.10380404

[4] D. C. Marcello, S. Shiber, O. De Marco, J. Frank, G. C. Clayton, P. M. Motl, P. Diehl, and H. Kaiser, "Octo-tiger: a new, 3d hydrodynamic code for stellar mergers that uses hpx parallelization," *Monthly Notices of the Royal Astronomical Society*, vol. 504, no. 4, pp. 5345–5382, 2021.

[5] H. Kaiser, P. Diehl, A. S. Lemoine, B. A. Lelbach, P. Amini, A. Berge, J. Biddiscombe, S. R. Brandt, N. Gupta, T. Heller *et al.*, "HPX-the C++ standard library for parallelism and concurrency," *Journal of Open Source Software*, vol. 5, no. 53, p. 2352, 2020.

[6] M. Salaün, "Landlock: unprivileged access control," 2024, last accessed 07/17/2024. [Online]. Available: https://docs.kernel.org/userspace-api/landlock.html

[7] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral, "The Spack Package Manager: Bringing Order to HPC Software Chaos," ser. Supercomputing 2015 (SC'15), Austin, Texas, USA, November 15-20 2015, lLNL-CONF-669890. [Online]. Available: https://github.com/spack/spack

[8] D. Marcello *et al.*, "Octo-tiger: Astrophysics program simulating the evolution of star systems based on the fast multipole method on adaptive octrees," last accessed 07/15/2024. [Online]. Available: https://github.com/STEllAR-GROUP/octotiger

[9] G. Daiß, J. Yan, P. diehl, and C. Junghans, last accessed 07/18/2024. [Online]. Available: https://github.com/G-071/octotiger-spack
[10] P. Diehl, "Data: Locking down science gateways," May 2024. [Online]. Available: https://doi.org/10.5281/zenodo.11355929
[11] "The einstein toolkit," last accessed 07/15/2024. [Online]. Available: https://einsteintoolkit.org