Static Analysis of Cimplifier: A Solution for Least Privilege Containers

Aarthi Vishwanathan, Chaitra Prasad Niddodi, Sibin Mohan
Dept. of Computer Science and Engineering , PES University, India
Dept. of Computer Science, Information Trust Institute, University of Illinois at Urbana Champaign, USA

ENGINEERING AT ILLINOIS
INFORMATIONTRUST INSTITUTE

INTRODUCTION

Containers are lightweight virtualization environments that run applications packed together with their resources.
However, these containers are not conducive to several security tenets.

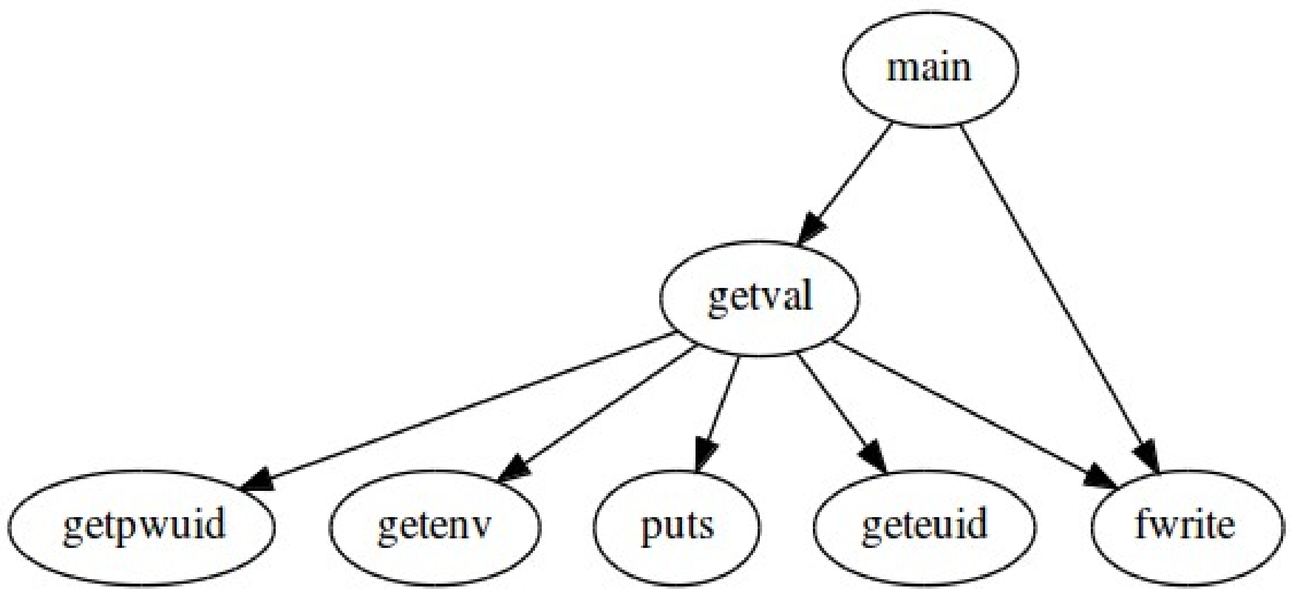
Here, we propose a tool called **Cimplifier.**

- We specifically look at **static analysis** to associate resources with each executable of a container.
- Given a container, resource set and user defined constraints, Cimplifier can produce containers having only the required set of resources.
- The original functionality of the container is preserved.

OBJECTIVES

- To analyze the usage of resources based on static analysis of system call logs and associate them with various executables.
- To understand the pros and cons of performing static analysis over dynamic analysis for resource identification.
- To use Cimplifier for partitioning a container and associating resources with the components of a partition.

APPROACH AND WORKFLOW



- For performing resource identification we use the **cflow** tool provided by the GNU Compiler Collection.
- By analyzing system call logs and the call graph, we come up with the set of dependencies and the read set and write set of resources associated with each executable of the container C, denoted as *R(C)* and *W(C)*.
- **strace** (for dynamic analysis) is combined with the output of cflow – to trace any undetected system calls and increase the code coverage.
- This output is given to Cimplifier, along with the user constraints, such as $(e_i, e_j) \in UC_+$
- Cimplifier slims down the containers , performs partitioning and gluing of the resultant containers.

OBSERVATIONS AND RESULTS

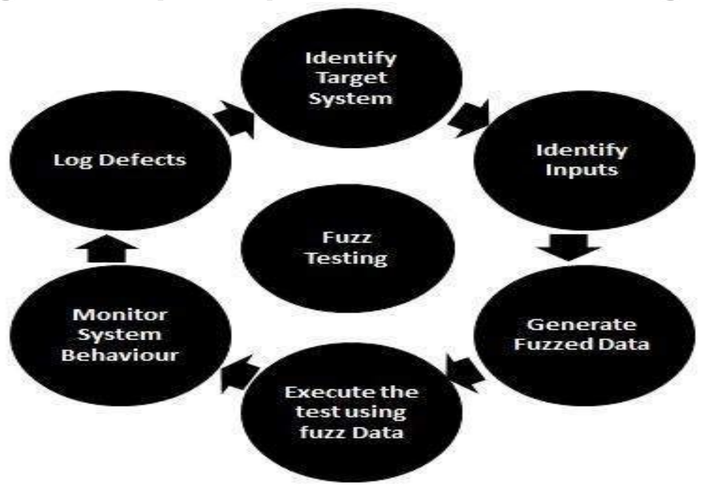
We ran Cimplifier on several single and multi stack application containers and noted the following :

Container	Original size	Resultant size	Size reduction
nginx	133 MB	5 MB	96.00%
python	199 MB	25 MB	87.00%
appcontainers/mediawiki	576 MB	270 MB	53.00%
eugeneware/docker-wordpress-nginx	602 MB	215 MB	64.00%

- Since there is better code coverage in static analysis, Cimplifier results in slightly better reduction in container size as compared to dynamic analysis alone.
- For multi-stack applications, as the executables can be strewn across multiple shared object libraries , there is greater memory overhead and hence, reduction in size with static analysis is not very efficient.

FUTURE WORK

- To ensure better coverage and to have most of the libraries along with the executables, users can use techniques like fuzzing where Cimplifier can be run in testing and pre-production stages.



- It will be useful to develop tools and workflows for debugging slimmed containers in deployment.