Maseeh College of Engineering and Computer Science

PORTLAND STATE UNIVERSITY

Functional Specification Standard

Project: Clang Randstruct
Author(s): Tim Pugh
Date: Nov 30, 2018

Contents

1. Introduction

1.1. Summary

randstruct is a GCC compiler plugin that was ported from grsecurity to upstream. This randomizes the layout of manually/automatically selected C structures. This makes flaw exploitation less deterministic, requiring significantly more flaws before an attacker can detect and target the layout of sensitive kernel structures in memory. Kees Cook, our sponsor, wants this functionality to be made usable in Clang/LLVM.

1.2. Requirements

1. **Full Randomization**
    a. All structures marked with "__randomize_layout" have their field positions randomized, including bit fields.
        i. Reproducible random number (same seed needs to be saved as part of the build)
        ii. Programs report field location manually (structure layout in gcc happens before plugin invocation)

2. **Performance-sensitive randomization (possible stretch)**
. Best-effort limit randomization to cache-line (64 byte) size regions, keep adjacent bit-fields together.
3. **Automatic structure selection (possible stretch)**
. Find structures that should be automatically selected (for example, structures of entirely function pointers), disabled with "__no_randomize_layout".
4. **Regression tests**
. Since the randomization must be stable from source-to-source, the randomization seed needs to be externally recorded (i.e. it is a build artifact). Regression tests to check all the corner cases should be built and included in the implementation (i.e. hooked to the standard LLVM/Clang regression tests)
5. **Publish this, regardless of upstream acceptance**
6. **Goal: upstream in LLVM/Clang (stretch)**
   Submit work upstream for review
   Implement changes based on feedback
   Repeat until accepted into LLVM/Clang upstream
7. **Kees Cook -- possibly assist in finding right person to ask a question to in the event of us getting blocked**

1.3. Numbers

The number of users of the randomization feature in Clang is unknown at this time and would be hard to estimate. Once implemented and approved by the upstream developers, many people will have the code at their disposal. As this is for an addition to a popular open source project, the number of users stands to be from 0 to millions. The usage among the users who know of the randomization mechanism, choose to use the randomization mechanism, and

their daily usage is unknown.

Provided users do intend to use the randomization features, we will have a method of turning on or off the feature for developers so that compilation times are not slower. The expected randomization is expected to incur a minimal overhead cost on small projects, and on larger ones it may prove large.

1.4. Existing System

Currently there is no system we are intending to replace, so much as providing an alternative too. The GNU Compiler Collection (GCC) is the alternative open source compiler that currently has a randomization feature implemented within it. The author of that system is our product stakeholder who is scheduled to receive the product of our implementation into Clang, Kees Cook.

1.5. Terminology

Clang – Compiler Front End for LLVM

GCC – The GNU Compiler Collection, C compiler sponsored by GNU

GNU – GNU not Unix

Randstruct – Randomization of struct fields, project name

Upstream – The “headwaters” of any forked project

Compiler – A program that takes source code and translates it to a target language

1.6. References

https://clang.llvm.org/docs/

https://clang.llvm.org/doxygen/

2. Functional Description

Use Cases

The Clang-Randstruct team will create a randomization mechanism to change the fields in structs for the Clang compiler front end. After it is created

developers will be able to use our mechanism by labelling structs with "__randomize_layout” and have their field positions randomized, including bit fields. This capability will be the base of other use cases, such as the performance mode and auto selection.

User Community

The user community of our randomization feature within clang varies on a per use case. The main consumers of our implementation are expected to be within the information security community, systems engineers, compiler engineers and others who need a hardening their software systems.

Error Handling

Errors in our system will be handled via extensive testing, debugging and iterative patches to our software. As we are contributing to an open source project with many people who may take an interest in what we are implementing, it’s reasonable to believe any errors overlooked may be captured and submitted to us via our software repository.

We will be leveraging Clangs own set of tools for testing our software as well as static code analysers, checking for memory leaks, security analysers, license analysers and using compilation flags to catch warnings.

Errors within our system must not take place or it may halt the compilation of code and prevent acceptance upstream. This would not be acceptable, and errors really cannot be allowed, although future development on languages may cause errors in the future, in which case we fully expect the Clang community to patch our work. Changes to the Clang compiler may also cause errors, but this too would be outside our control and would be unforeseeable.

Security

Our project is intended to harden the security of other software. Possible abuses to our system may be possible via a reverse engineering of the randomization seed for a run of a user’s software. In the event someone circumvents the system, a user would simply need to restart their system. The likelihood of an abuse of our system is minimal due to needing to reverse the clang code base which would not be trivial to accomplish.

Help

Within the duration of the project, help will be provided when users request to test the functionality. We will provide all documentation and source code and will be using Doxygen to generate documentation for the code that is written. This is standard of the clang community and we intend to fulfil this obligation as we understand it to be a requirement for being accepted upstream.

As the project is limited in duration, most help through the 3rd week of March ranges from none to extensive. At the very least, one software engineer will remain within the community and be available to help with any issues that may arise. All other issues may be directed to the clang developers who are reachable via IRC, group mailing list, and email.

Interfaces

User

The use of our system will take place at the command line via switches to the clang compiler.

Software

Software developers may mark their code for use within our system by labelling their structs with "__randomize_layout".

Boundary Conditions

Our software should not incur any boundary conditions due to the size of any code going through the randomization process.

Constraints.

Technical constraints do exist in the development of the randomization implementation. The domain knowledge for a compiler is large and will be challenging to overcome.

Several unique constraints do exist for our project, namely getting it into upstream may prove very difficult. This is outside of our control, but we intend to fork from Clang in the event it isn't so other developers can continue to make additions as well as team members when our contract expires.

A scheduling constraint also exists where we are obligated to work on the system only up until March 18th, 2019. After this time, we fully expect contributors outside of our project to contribute as we will be publishing our work to the open source and LLVM/Clang community and figuratively "putting

the word out". We intend to post to the developer mailing list, and LLVMWeekly.com, a weekly newsletter for LLVM/Clang published weekly. We also intend to continue our dialog with past contacts who have aided in development.

Platforms

Tests will be conducted on x86 Linux platforms and if time allows tested on all other platforms. In most instances, the platforms that clang supports now should be supported by our system as well.

Internationalisation

No internationalization is expected in our software. The Doxygen documentation will be available if accepted into upstream, in which case a web browsers translation software will suffice as a stand-in. This is outside the scope of our project.

Performance

Capacity: Our software should not incur any problems with capacity that don't already exist within Clang currently.

Response times: We will be implementing a performance sensitive randomization concurrently. Further testing will be needed to see the performance overhead our code pays to randomization with compiling the code but will constant and only should be microseconds indifference from any struct. A performance hit will be expected but should not differ from struct to struct or will be inconsequential.

Portability

Although we may only be supporting one platform initially, we almost certainly will want to be able to port developments to other platforms.

Expandability

Expandability may be carried out freely under our intended software license. Anyone will be free to modify our code.

Customisation

No customization is necessary nor supported besides being able to seed the randomization feature by the user.

Support & Maintenance

There will be tests implemented during development to ensure that the randomization features are correct. These will be included if desired by the upstream community but will be developed regardless of concurrently with the system. Future maintenance by the team will be on a case by case basis but generally deferred to the LLVM/Clang development community and repository maintainers.

Configuration Management

Management of the system will be unique as our project will be implemented onto a pre-existing open source project. We intend to license the software

with the same license as LLVM/Clang so that users developing on Clang can expect the same restrictions for modifying the code. All future configuration will be at the will of the Clang developers and repository maintainers.

Documentation

We will be using the documentation generation tool Doxygen as this is what is currently used by Clang. All documentation will be within the source code, and if accepted to upstream, there should be a mechanism in place that will publish the documentation online. We will be documenting all source material.