ARTIFACT EVALUATED
usenix ASSOCIATION
AVAILABLE

USENIX'23 Artifact Appendix: Not All Data are Created Equal: Data and Pointer Prioritization for Scalable Protection Against Data-Oriented Attacks

Salman Ahmed, Hans Liljestrand, Hani Jamjoom, Matthew Hicks, N. Asokan, Danfeng (Daphne) Yao

A Artifact Appendix

A.1 Abstract

This artifact provides a comprehensive guide on installing and utilizing our proposed Data and Pointer Prioritization (DPP) framework. The DPP framework incorporates rule-based heuristics to automatically identify and prioritize/rank sensitive memory objects from an application. Within this artifact, we outline the necessary prerequisites, requirements, and software dependencies for DPP, along with detailed instructions on accessing, setting up, and installing the framework. Additionally, we delve into the usage of the DPP framework, specifically focusing on prioritizing sensitive data objects through a straightforward program.

A.2 Description & Requirements

The source code of DPP consists of a set of LLVM analysis passes and modifications to the AddressSanitizer's (ASan) instrumentation mechanism. To generate the data-flow graph, DPP utilizes the SVF tool (available at `https://github.com/SVF-tools/SVF`). We have made changes to LLVM's build script (CMakeFiles.txt) to include SVF's source as an in-tree build (as a library) during the compilation of the LLVM source code. For ease of use, we provide the build script (build.sh). Additionally, we have also modified SVF's build script and included a customized version of SVF in our repository. It's important to note that compiling and building the LLVM source code requires CMake and the Ninja build systems as prerequisites.

Regarding the datasets, most of them, such as the Juliet Test Suite and the Linux Flaw Project, are publicly available. Additionally, the source codes of other applications used in our evaluation can also be accessed publicly. However, to simplify access and ensure convenience, we have included all of these datasets in our repository.

A.2.1 Security, privacy, and ethical concerns

No destructive steps are taken or no security mechanism are disabled during the build process of DPP. One just needs to install a compatible version (>= 3.13.4) of CMake.

A.2.2 How to access

The source code of DPP is available publicly on GitHub at `https://github.com/salmanyam/DPP` with commit 53cbccb. The full URL is `https://github.com/salmanyam/DPP/tree/53cbccb6e6eaab6eaabbb06ea21fd31dd83e6eff`.

A.2.3 Hardware dependencies

None

A.2.4 Software dependencies

To compile and build the LLVM source code containing DPP's passes, we use Ubuntu 18.04. You will need CMake version 3.13.4 or higher to successfully compile and build the LLVM source code. Additionally, we suggest using the Ninja build system for this process. Ideally, on a system with all the necessary prerequisites, including Ninja and a compatible CMake version, the build process should proceed smoothly without any complications. It's worth noting that while the scripts have been tested on Ubuntu 18.04, they should also be compatible with the latest Ubuntu distributions.

A.2.5 Benchmarks

All the datasets and source codes that have been used in our evaluation are publicly available. However, we have provided the datasets and source code in our repository.

- Juliet Test Suite: `https://github.com/salmanyam/dpp-data/tree/main/juliet-test-suite`
- Linux Flaw Project: `https://github.com/salmanyam/dpp-data/tree/main/linux_flaw_project`
- Other applications' source code: `https://github.com/salmanyam/dpp-data/tree/main/src`
- Besides, we provide the LLVM IR files in `https://github.com/salmanyam/dpp-data/tree/main/IRx86`

A.3 Set-up

To replicate the evaluation environment for DPP, we recommend setting up an Ubuntu 18.04 distribution. To ensure a compatible CMake version is installed, we provide a convenient script called `install_cmake.sh`. After running this script, it is necessary to exit and restart the terminal to apply the installation changes effectively. Additionally, we offer another script, `prerequisites.sh`, which installs all the necessary prerequisites. If you have a fresh installation of the Ubuntu 18.04 distribution, please follow these steps from the root directory of our repository:

```
 $ ./prerequisites.sh
 $ ./install_cmake.sh
```

By following these steps, you can quickly set up the required environment for DPP and ensure all dependencies are properly installed.

A.3.1 Installation

To compile and build DPP along with the LLVM source, one needs to issue the build script (`build.sh`) provided in our repository. This script will do an in-source compilation of SVF and build LLVM binaries (`clang`, `opt`, `llvm-ar`, `lld`, etc) under `dpp-llvm/build/bin`.

A.3.2 Basic Test

To show simple prioritization results, we provide a simple program (`dpp-data/example/example.c`) and its LLVM IR (`dpp-data/example/example.opt`). The following commands give simple prioritization results.

```
$ LLVM_DIR=${PWD}/dpp-llvm/build/bin
$ ${LLVM_DIR}/opt -S -passes="print-dpp-global"
     --dpp-rule="all" -disable-output < ${PWD}/
    dpp-data/example/example.opt
```

The commands run all rules and prioritize/ranks the data objects in the simple program. The output of the command is the following:

```
######## SUMMARY: 4 data objects #############
AddrVFGNode ID: 17 AddrPE: [34<--35]
   %9 = call noalias i8* @malloc(i64 %8) #6, !
       dbg !25 { ln: 8 cl: 25 fl: example.c } 4
        10
----------------------------------------------
AddrVFGNode ID: 15 AddrPE: [22<--23]
   %4 = alloca i8*, align 8 { ln: 8 fl: example.
       c } 2 10
----------------------------------------------
AddrVFGNode ID: 11 AddrPE: [6<--7]
 @stdin = external dso_local global %struct.
     _IO_FILE*, align 8 { Glob } 1 1
----------------------------------------------
AddrVFGNode ID: 14 AddrPE: [20<--21]
   %3 = alloca [10 x i8], align 1 { ln: 6 fl:
       example.c } 1 0
----------------------------------------------
```

As can be seen from the output, there are four data objects prioritized by DPP for the simple program. Since the simple program is small and most data objects are input-dependent, almost all the data objects have been prioritized.

The following command is used to run the prioritization using a single rule:

```
$ LLVM_DIR=${PWD}/dpp-llvm/build/bin
$ ${LLVM_DIR}/opt -S -passes="print-dpp-global"
     --dpp-rule="rule9" -disable-output < ${PWD
    }/dpp-data/example/example.opt
```

The output of the above command is as follows:

```
AddrVFGNode ID: 17 AddrPE: [34<--35]
   %9 = call noalias i8* @malloc(i64 %8) #6, !
       dbg !25 { ln: 8 cl: 25 fl: example.c }
----------------------------------------------
```

In addition to this simple program, we have provided many LLVM IR files for real-world applications in https://github.com/salmanyam/dpp-data/tree/main/IRx86. We can use the abovementioned command to obtain the prioritized data objects by changing the input to those commands. For example, the following command takes the IR file of wuftpd and runs the prioritization using all rules.

```
$ LLVM_DIR=${PWD}/dpp-llvm/build/bin
$ ${LLVM_DIR}/opt -S -passes="print-dpp-global"
     --dpp-rule="all" -disable-output < ${PWD}/
    dpp-data/IRx86/wuftpd-2.6.0.bc
```