Saint: Simple Static Taint Analysis Tool User's manual

Dipl.-Inf. Xavier NOUMBISSI NOUNDOU
xavier.noumbis@gmail.com

August 30, 2015

Contents

List of Tables

1 Introduction

Businesses increasingly use software. This is even more relevant for companies relying on e-commerce. However, software is error-prone and contain several bugs. Security bugs are one of the major problems faced by companies today. In the worst case, security bugs enable unauthorized users to gain full control of an application.

My PhD thesis introduces the concept of *tainted paths* and describes techniques and algorithms to compute them in any imperative programming language that uses pointers (C, C++, Java, etc.). I implemented these algorithms in SAINT.

SAINT does not require the developer to annotate the program under analysis. SAINT implements a flow-sensitive, interprocedural and context-sensitive analysis that computes tainted paths in C programs at compile-time.

2 Installation Instructions

This section explains how to install SAINT on a "Linux" machine. We haven't tested SAINT on a "Windows" or "Mac OS" machine, but the installation should follow similar steps.

2.1 Required Software

This section enumerates all software that you need to run SAINT.

- SAINT: https://github.com/xaviernoumbis/saint.git
- The compiler infrastructure `LLVM, version 3.3`: http://llvm.org
- The DSA pointer analysis `poolalloc`: https://github.com/llvm-mirror/poolalloc.git

2.2 Environment Variables

Table 1 that shows all environment variables that you have to define and export in order to successfully run SAINT.

✓ You define and export an environment variable ENV_VAR by writing the following commands in your "`$HOME/.bashrc`" file:

```
ENV_VAR=path_to_folder
export ENV_VAR
```

Environment variables	Description
SAINT_HOME	`SAINT` home folder (e.g. /home/user/saint)
SAINT_BIN	`SAINT` binaries folder (e.g. $SAINT_HOME/bin)
LLVM_HOME	`llvm` home folder (e.g. /home/user/llvm)
LLVM_LIB	`llvm` compiled libraries folder (e.g. $LLVM_HOME/build/Release+Asserts/lib)
LLVM_BIN	`llvm` compiled binaries (e.g. $LLVM_HOME/build/Release+Asserts/bin)
POOLALLOC	`poolalloc` home folder (e.g. /home/user/poolalloc)
CLANGLLVM_BIN	`clang+llvm` binaries' folder (e.g. /home/user/clang+llvm/bin)

Table 1: Table with all environment variables required to install and use `SAINT`

3 Folder Structure

The following folders constitute `SAINT`'s directory structure:

1. `bin`: folder with the scripts:
   - `saint-gen-ir.sh`: generates the LLVM IR for code analyzed by `SAINT`.
   - `saint-run-llvm-opt.sh`: runs LLVM (`opt` binary) with `SAINT` as plugin.
   - `saint-configure.sh`: configures and compiles `poolalloc` and `LLVM` for `SAINT`.
2. `benchmarks`: folder with sample scripts to run `SAINT`.
3. `cfg`: folder with source, sink, and sanitizer configuration files
4. `projects`: folder with sample projects.
5. `doc`: folder with the manual.
6. `src`: folder with all `C++` source files, and `Bash` scripts to compile and run `SAINT`.

4 How to Configure "`clang+llvm`" for use with SAINT

a) Download and unpack `clang+llvm, version 3.3`.

b) Add the `bin` folder to your environment variable PATH.
   ✓ For instance by adding the following line in your file "`$HOME/.bashrc`"

```
PATH=$PATH:$CLANGLLVM_BIN
export PATH
```

-

5 How to Configure "LLVM" and "poolalloc" for use with SAINT

a) Open the file "$LLVM_HOME/lib/Analysis/Makefile" and append the string "saint" to the "DIRS" variable. Following is an excerpt of the file.

```
##===- lib/Analysis/Makefile ------------------------------*- Makefile -*-===##
#
#                     The LLVM Compiler Infrastructure
#
# This file is distributed under the University of Illinois Open Source
# License. See LICENSE.TXT for details.
#
##===----------------------------------------------------------------------===##

LEVEL = ../..
LIBRARYNAME = LLVMAnalysis
DIRS = IPA saint
BUILD_ARCHIVE = 1

include $(LEVEL)/Makefile.common
```

b) Run the script `saint-configure.sh`.

6 How to Compile and Run SAINT

6.1 How to Compile SAINT

You need to execute the command "make -f Makefile.saint" within the folder "$LLVM_HOME/lib/Analysis/saint" to compile SAINT.

SAINT also gets compiled when you run it using the Bash script `saint-run-llvm-opt.sh`.

6.2 Configuration Files

Configuration files are found in the folder "$SAINT_HOME/cfg". There are three configuration files:

- `sources.cfg`: taint sources configuration file
  Each line of the file specifies a function name and an integer "x". "x" is the

argument of the function that is tainted.
If "`x`" is zero (0), then it is the return value of the function that is tainted.

```
fopen,0
fgets,1
```

The previous lines specify that `fopen` returns a tainted value, and that `fgets` taints its first argument.

- `sinks.cfg`: taint sinks configuration file
  Each line of the file specifies a function name and an integer "`y`". "`y`" is the argument of the function that must not received a tainted value.
  "`y`" is never equal to zero (0).

```
sprintf,1
```

The previous line denotes that `sprintf` must not received a tainted value as its first argument (i.e. `sprintf` is sensible function.

- `sanitizers.cfg`: sanitizers configuration file
  SAINT doesn't yet implement this functionality.

6.3 How to Run SAINT

Among others, SAINT source folder contains the following two important `Bash` scripts:

- `saint-gen-ir.sh`: this script is used to generate and merge `llvm` intermediate representation (IR) files.
- `saint-run-llvm-opt.sh`: this script is used to run the analysis of SAINT on the program under analysis.

We encourage users to look at the sample scripts in the folders "`projects`" and "`benchmarks`" to learn how to use `saint-gen-ir.sh` and `saint-run-llvm-opt.sh`.

6.4 How to Get Debug Messages from SAINT

There are different kinds of debug messages SAINT outputs. You choose the type of debug messages using the option "`e`" of the script `saint-run-llvm-opt.sh`. Table 2 shows for each debug message which parameter to use with option "`e`". An example is:

```
saint-run-llvm-opt.sh -o "$CLANGLLVM_BIN/opt" -s -e "saint-table" \
  -t -p "Main sample" -i results/one.bc 2> results/analysis-result
```

Type of debug message	**Parameter for option "e" of script** `saint-run-llvm-opt.sh`
Tainted paths and summary table information	saint-table
Warnings	saint-warnings
Taint sink functions	saint-sinks
Taint source functions	saint-sources

Table 2: SAINT Debug Message Options

If option "`d`" of the script `saint-run-llvm-opt.sh` is used instead of "`e`", then all debugging information from LLVM and other used libraries is output. For instance:

```
saint-run-llvm-opt.sh -o "$CLANGLLVM_BIN/opt" -s -d \
  -t -p "Main sample" -i results/one.bc 2> results/analysis-result
```

Author's Biography

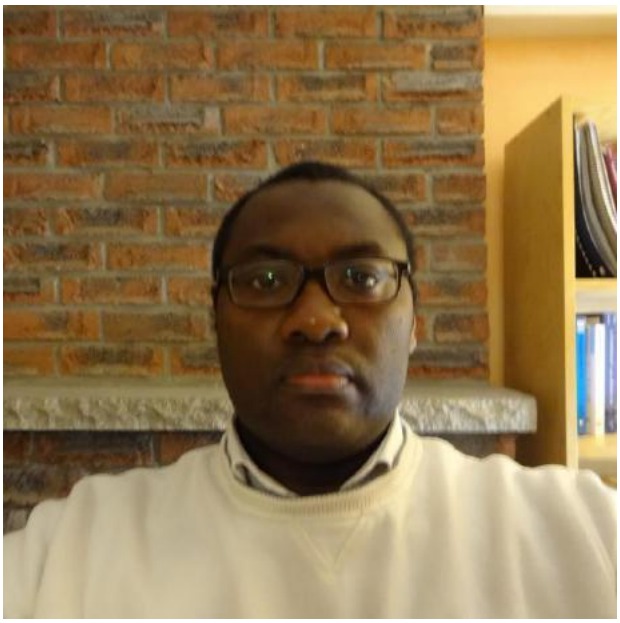

Figure 1: The author Xavier NOUMBISSI NOUNDOU

Xavier NOUMBISSI NOUNDOU is from Cameroon and holds the title Diplom-Informatiker (DIPL.-INF.) of the **University of Bremen** [1] in Germany. A DIPL.-INF. degree is roughly equivalent to a canadian Master's degree in Computer Science.

After his Diplom-Informatiker degree, he worked 18 months as Software Developer for **Siemens Healthcare** [2] in Erlangen (Germany).

After **Siemens Healthcare** , Xavier started his doctoral research in Program Analysis and Software Engineering in the **Watform Lab.** at the **University of Waterloo** [3] (Waterloo, Ontario, Canada).

From January 2012 to August 2012, Xavier worked in the Java (J9) Just–In–Time compilation team of the **IBM Toronto Lab.** in Markham (Ontario, Canada) as a graduate intern in compiler optimization.

Xavier is professionally proficient in the French, English and German languages.

For his DIPL.-INF. degree, Xavier worked on the automatic generation of test cases for reactive systems. The algorithms he developed are used by the German company

[1] http://www.uni-bremen.de
[2] http://www.healthcare.siemens.com
[3] http://www.uwaterloo.ca

Verified Systems International GmbH [4]. The title of his diplom-informatiker degree thesis was "`Statistical test cases generation for reactive systems`".

Xavier currently works on his PhD degree. His research focuses on static taint analysis which is a technique used to find *tainted data* in programs. *tainted data* are data that originates from user input and that have not been validated and sanitized.

Xavier is the creator of **SAINT** [5], which is a tool to perform static taint analysis on programs written in the C programming language.

[4] http://www.verified.de
[5] https://www.github.com/xaviernoumbis/saint