

```
#####
#
# Notices:
#
# Copyright (c) 2011 United States Government as represented by the
# Administrator of the National Aeronautics and Space Administration.
# All Rights Reserved.
#
# Disclaimers:
#
# No Warranty: THE SUBJECT SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY WARRANTY OF
# ANY KIND, EITHER EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED
# TO, ANY WARRANTY THAT THE SUBJECT SOFTWARE WILL CONFORM TO SPECIFICATIONS,
# ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE,
# OR FREEDOM FROM INFRINGEMENT, ANY WARRANTY THAT THE SUBJECT SOFTWARE WILL BE
# ERROR FREE, OR ANY WARRANTY THAT DOCUMENTATION, IF PROVIDED, WILL CONFORM TO
# THE SUBJECT SOFTWARE. THIS AGREEMENT DOES NOT, IN ANY MANNER, CONSTITUTE AN
# ENDORSEMENT BY GOVERNMENT AGENCY OR ANY PRIOR RECIPIENT OF ANY RESULTS,
# RESULTING DESIGNS, HARDWARE, SOFTWARE PRODUCTS OR ANY OTHER APPLICATIONS
# RESULTING FROM USE OF THE SUBJECT SOFTWARE. FURTHER, GOVERNMENT AGENCY
# DISCLAIMS ALL WARRANTIES AND LIABILITIES REGARDING THIRD-PARTY SOFTWARE,
# IF PRESENT IN THE ORIGINAL SOFTWARE, AND DISTRIBUTES IT "AS IS."
#
# Waiver and Indemnity: RECIPIENT AGREES TO WAIVE ANY AND ALL CLAIMS AGAINST
# THE UNITED STATES GOVERNMENT, ITS CONTRACTORS AND SUBCONTRACTORS, AS WELL
# AS ANY PRIOR RECIPIENT. IF RECIPIENT'S USE OF THE SUBJECT SOFTWARE RESULTS
# IN ANY LIABILITIES, DEMANDS, DAMAGES, EXPENSES OR LOSSES ARISING FROM SUCH
# USE, INCLUDING ANY DAMAGES FROM PRODUCTS BASED ON, OR RESULTING FROM,
# RECIPIENT'S USE OF THE SUBJECT SOFTWARE, RECIPIENT SHALL INDEMNIFY AND HOLD
# HARMLESS THE UNITED STATES GOVERNMENT, ITS CONTRACTORS AND SUBCONTRACTORS,
# AS WELL AS ANY PRIOR RECIPIENT, TO THE EXTENT PERMITTED BY LAW.
# RECIPIENT'S SOLE REMEDY FOR ANY SUCH MATTER SHALL BE THE IMMEDIATE,
# UNILATERAL TERMINATION OF THIS AGREEMENT.
#
#####/
```

=====  
PREREQUISITES  
=====

This document provides guidelines for installing IKOS on Mac OS X. The installation process described here has been successfully tested on Mac OS 10.8.

General Rules of Installation:  
-----

IKOS requires a number of third-party packages (all of which are open source) to be installed on your Mac. The easiest way to install these packages is to use MacPorts. Although it is possible to install them manually, we only give installation instructions using MacPorts. If you don't have MacPorts installed on your computer, you can check the MacPorts website for download instructions.

Installing MacPorts:  
-----

Before installing any package, make sure that your version of MacPorts is up-to-date by typing the following command in a shell window:

```
> sudo port selfupdate
```

Note that using MacPorts requires having administrator privileges on your Mac. If you don't, you shall contact your system administrator for assistance.

Installing GCC 4.2:

-----  
IKOS won't compile properly with a version of GCC older than 4.2. Recent versions of gcc may also cause compilation issues. If you don't have GCC 4.2 installed on your Mac, you can install it as follows:

```
> sudo port install apple-gcc42
```

Note that you shall install the Apple version of GCC 4.2. The standard version will not work properly on Mac OS X. Then, you shall set this new version as the default GCC compiler:

```
> sudo port select --set gcc apple-gcc42
```

Finally, you shall update the links in /usr/bin so that the right version of the compiler will be used during builds:

```
> sudo ln -fs /opt/local/bin/gcc-apple-4.2 /usr/bin/gcc  
> sudo ln -fs /opt/local/bin/g++-apple-4.2 /usr/bin/g++
```

Installing BOOST:

-----  
IKOS makes extensive use of the BOOST library. To install it, just type the following command in a shell:

```
> sudo port install boost
```

Note that the version of BOOST selected shall be 1.47.0 or later.

Installing GMP:

-----  
IKOS uses the GMP library for arbitrary precision arithmetic. To install GMP and the corresponding C++ wrappers, just type the following command in a shell:

```
> sudo port install gmp
```

Note that you shall install version 5 or later.

Installing SQLite:

-----  
The analyses built with IKOS use SQLite for performing offline data management. It can be installed using the following shell command:

```
> sudo port install sqlite3
```

Note that version 3 of SQLite is required.

Installing wget:

-----

IKOS automatically downloads the LLVM front-end from the official distribution website. In order to enable this feature you need to have wget installed on your machine. To do so, just type the following command in a shell:

```
> sudo port install wget
```

```
=====
  SETUP
=====
```

The IKOS distribution comes as a compressed tarball named `ikos_arbos.xx.yy.tar.gz`, where `xx.yy` is a version number. Unpack the distribution somewhere in your home directory and set the value of the environment variable `IKOS_INSTALL` to the absolute path of the distribution. For example, if you've unpacked the distribution in the directory `/Users/myself/tools/ikos_arbos.xx.yy`, then you shall add the following command to your profile:

```
export IKOS_INSTALL=/Users/myself/tools/ikos_arbos.xx.yy
```

Also add the following environment variable definitions:

```
export BOOST_INSTALL=/opt/local/include
export GMP_INSTALL=/opt/local/lib
```

You might also want to update your `PATH` variable as follows:

```
export PATH="$IKOS_INSTALL/bin:\
    $IKOS_INSTALL/llvm-gcc/bin:\
    $IKOS_INSTALL/llvm-src/Release/bin:\
    $PATH"
```

To finish the install, please go to the directory containing the unpacked IKOS distribution and type:

```
> make all
```

This will download the LLVM front-end, install it and compile the IKOS static analyzer.

If for some reason the installation of IKOS fails after installing LLVM, typing 'make all' again will start over the installation from the beginning. In order to skip the installation of LLVM, simply type:

```
> make ikos
```

Instead of typing 'make all', you can perform the installation in two steps with the following commands:

```
> make llvm
> make ikos
```

```
=====
USAGE
=====
```

Compiling a C program with LLVM:

-----

This requires modifying the Makefile used to build the program so that the compiler tools invoked are those provided by LLVM. This usually amounts to changing the settings for the Makefile variables CC, LD and AR. This process is illustrated in the directory 'example' located under the IKOS installation directory. There, you can find a simple program made of three files 'main.c', 'f1.c' and 'f2.c'. The files 'f1.c' and 'f2.c' are compiled separately and placed in a library, which is then linked with the main C program. Here is what the original Makefile looks like:

```
CC = gcc -c
LD = gcc
AR = ar

LIB_FILES = f1.o f2.o

all: main.o lib.a
    $(LD) -o example main.o lib.a

%.o: %.c
    $(CC) -o $@ $<

lib.a: $(LIB_FILES)
    $(AR) rs $@ $(LIB_FILES)

clean:
    rm -f *.o lib.a example
```

Modifying this Makefile so that it can be compiled by the LLVM front-end only requires changing the settings of the compiler variables as follows:

```
LLVM_INSTALL=${IKOS_INSTALL}/llvm-src/Release/bin
LLVM_GCC_INSTALL=${IKOS_INSTALL}/llvm-gcc/bin

CC = $(LLVM_GCC_INSTALL)/llvm-gcc -emit-llvm -fno-inline -c -g
LD = $(LLVM_INSTALL)/llvm-ld -link-as-library -disable-inlining -disable-opt
AR = $(LLVM_INSTALL)/llvm-ar
```

The rest of the Makefile is unchanged. The program can then be built using LLVM by typing the following command in a shell:

```
> make -f Makefile.llvm
```

The binary 'example' now contains LLVM machine code that can be processed by the IKOS static analyzer. The settings listed above are generic and can be used in any Makefile that uses CC, LD and AR. If the Makefile directly invokes the compiler tools, each invocation of gcc, ld or ar shall be manually modified.

Running the buffer-overflow static analyzer:  
-----

The IKOS static analyzer that checks for buffer overflows is named 'boa' and is located in the 'bin' directory of the IKOS distribution. The analysis can be run in two modes:

- The intraprocedural mode (command-line option '-intra'), which ignores function call contexts, is fast but imprecise.
- The interprocedural mode (command-line option '-inter') takes into account function call contexts and is much more precise. It is also more costly computationally and cannot handle recursive programs.

To run the analysis in intraprocedural mode on the example, just type the following command in a shell:

```
> boa -intra example
```

The results of the analysis are stored in an SQLite database named 'output.db'. To browse the results, just use the SQLite shell:

```
> sqlite3 output.db
```

The results are stored in a table named 'boa\_results' with the following structure:

```
sqlite> .schema
CREATE TABLE boa_results(safety_check, file, line, status);
CREATE INDEX boa_results_index_1 ON boa_results(safety_check);
CREATE INDEX boa_results_index_2 ON boa_results(file);
CREATE INDEX boa_results_index_3 ON boa_results(line);
```

The results for the example are:

```
sqlite> select * from boa_results;
overflow|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f1.c|6|ok
underflow|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f1.c|6|ok
overflow|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f1.c|8|error
underflow|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f1.c|8|ok
overflow|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|6|warning
underflow|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|6|ok
overflow|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|8|warning
underflow|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|8|ok
```

The column 'safety\_check' describes the type of buffer access checked: 'overflow' for accessing an element past the end of a memory block and 'underflow' for an access with a negative offset. The 'file' and 'line' give the location of the operation checked in the original source code. The column 'status' describes the conclusion of the static analyzer on the buffer access checked:

- 'ok' means that the buffer access is safe for all execution contexts;
- 'error' means that the buffer access always results into an error, regardless of the execution context;
- 'warning' may mean two things: (1) the operation results into an error for some execution contexts but not other, or (2) the static analyzer did not have enough information to conclude, because either the program does not provide enough information (check dependent on the value of an external input

for example) or the static analysis algorithms are not powerful enough;  
- 'unreachable' (not listed here) means that the code in which the buffer operation is located is never executed (dead code).

For example, all array accesses inside the loop of function f1 are safe, whereas the operation upon loop exit tries to access an element past the end of the array. All array operations in function f2 are flagged as potentially unsafe, since some call contexts lead to a buffer overflow and some other are safe. However, the analysis is not precise enough to tell us what execution contexts lead to an error. These result can be improved upon using the interprocedural analysis, which can be launched by the following command:

```
> boa -inter example
```

The results are stored in the same table 'boa\_results', which has been augmented with a column 'context' giving the context in which a function is called:

```
> sqlite3 output.db
SQLite version 3.7.12 2012-04-03 19:43:07
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE boa_results(safety_check, context, file, line, status);
CREATE INDEX boa_results_index_1 ON boa_results(safety_check);
CREATE INDEX boa_results_index_2 ON boa_results(file);
CREATE INDEX boa_results_index_3 ON boa_results(line);
```

A call context is a sequence of call sites of the form f@l, where f is a function name and l is a line number. The results of the analysis for the example are the following:

```
sqlite> select * from boa_results;
overflow|.:.main@6|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f1.c|6|ok
underflow|.:.main@6|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f1.c|6|ok
overflow|.:.main@6|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f1.c|8|error
underflow|.:.main@6|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f1.c|8|ok
overflow|.:.main@7|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|6|ok
underflow|.:.main@7|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|6|ok
overflow|.:.main@7|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|8|ok
underflow|.:.main@7|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|8|ok
overflow|.:.main@8|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|6|warning
underflow|.:.main@8|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|6|ok
overflow|.:.main@8|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|8|error
underflow|.:.main@8|/Users/ajvenet/test-release/ikos_arbos.0.1/example/f2.c|8|ok
```

The static analyzer has been able distinguish between the two calls to f2 and give precise answers in each case. The only remaining warning in the second call to function f2 cannot be further refined as the array operation inside the loop is safe for the first iterations but ultimately results into an error.