==========
 PREREQUISITES
==========

This document provides guidelines for installing IKOS on a Linux system. The installation process described here has been successfully tested on OpenSUSE 11.4 and Ubuntu 12.

General Rules of Installation:

IKOS requires a number of third-party packages (all of which are open source) to be installed on your Linux system. The packages requiring no modification are likely already installed as part of the Linux OS installation. One should confirm that the version numbers of the pre-installed packages are appropriate for IKOS as will be stated below. Other packages will require manual compiling and installation. Installation of the packages is most easily accomplished with administrative privileges. Pre-installed packages will likely be found under /usr with some subirectories including the architectural bit size, e.g. 32 or 64, in the name. etc. ./lib64. Manually installed

packages will generally, by default, be installed under /usr/local.

Be sure to only use the latest stable release of a package. Avoid installing betas or from factory development trunks. All linux distributions have a chosen package management approach. The two major distributions - Red Hat (Fedora) uses YUM and RPM files and Ubuntu has its highly integrated Software Center that uses files with the 'deb' extension (originally the Debian software package format). OpenSUSE uses the YaST driven by Zypp and uses RPM packages and supports YUM dB format. When practical, installation through the distribution's software package manager is recommended but installation from a TGZ (tar,gzip) package is also acceptable but requires future maintenance be done manually.

Installing GCC 4.2:

IKOS will not compile properly with a version of GCC older than 4.2. Later versions are known to cause compilation problems too. Hence, installing GCC 4.2 is highly recommended. Check the version number of the installed GCC as follows:

> gcc --version

Versions of GCC that are confirmed able to compile IKOS utilities: GCC 4.2.x, 4.3.x, 4.4.x and 4.5.x. If your system has a later version installed, it is recommended that one of these releases be installed in /usr/local and LIB/PATH point to the directories appropriately. Note: compiling an additional GCC requires installation of various libraries. Be sure to include the development/source packages of the libraries which will include, e.g. essential header files.

Installation, via command line for different Linux distributions is as follows:

Fedora:
> sudo yum install gcc-g++

Ubuntu:
> sudo apt-get install gcc-g++

OpenSUSE:
> sudo yast install gcc-g++

Installing BOOST:

IKOS makes extensive use of the BOOST library. Major Linux distribution maintain an up to date release of BOOST that can be readily installed using the package manager. To install it, type the following command in a shell window:

> sudo <yum/apt-get/yast> install boost

Note that the version of BOOST must be 1.47.0 or later.

Note that the correct package on Ubuntu 12 is libboost-all-dev.

Note also that for the current stable version of Ubuntu (12.04) the command 'sudo apt-get install libboost-all-dev' might install the version 1.46.0. If you encounter later during the compilation process some error related to BOOST we recommend to download a newer version from the BOOST webpage http://www.boost.org/users/download/

Installing SQLite:

IKOS uses SQLite as an infrastructure for performing offline data management. SQLite is a standard package in practically all Linux distributions. Insure that version 3 or later is installed in your Linux system.

Note: Use apt-get install sqlite3 for Ubuntu.

Installing GMP:

IKOS uses the Gnu Multi-Precision Library (GMP) for arbitrary precision arithmetic. GMP is commonly maintained as a package by the major Linux distributions. If not available for your Linux distribution, then go to the GMP web site and download the latest stable version in a familiar format, e.g. tar.gz. Install gmp and the GMP C++ bindings using a software manager from one of the major linux distributions. Both 64 and 32 bit systems are supported by following two packages.

> sudo <yum/apt-get/yast> install gmp-devel
> sudo <yum/apt-get/yast> install libgmpxx4

Note that you shall install version 5 or later.

Note: Use apt-get install libgmp-dev for Ubuntu (this includes libgmpxx4).

=========
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=/usr/include/boost

Note: if you needed to manually install BOOST from the official distribution (http://www.boost.org/users/download), just set BOOST_INSTALL to the root of

the BOOST install directory.

export GMP_INSTALL=/usr/lib

If the installation fails because it cannot find GMP you might also
try

'ls /lib/libgmp* /usr/lib/libgmp* /usr/share/lib/libgmp* /usr/local/lib/libgmp* \
 /usr/local/share/lib/libgmp* /opt/lib/libgmp* /opt/gmp/libgmp* /opt/local/lib/libgmp* \
 /usr/lib/x86_64-linux-gnu/libgmp* /usr/lib/i386-linux-gnu/libgmp* '

If you get a match, you can use that directory name as a value for
GMP_INSTALL. Be aware that you could get two matches, e.g., with
/usr/lib/x86_64-linux-gnu and /usr/lib/i386-linux-gnu if you have installed
both a 64-bits and 32-bits version of GMP in your machine. In that case, please
type the command 'uname -m' and choose the correct install.

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 chaging 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.