Version: 1.1.0

On this page

Hello, World!

Follow the instructions in this page to try Infer on a few small examples. You should be able to see the bugs reported by Infer, fix the bugs and run Infer again to check that they are not reported anymore. This should give you a first idea of how Infer works. See the User Guide for more details about the use of Infer.

All these examples can be found in the `infer/examples` directory distributed with Infer.

Hello world Java

Here is a simple Java example to illustrate Infer at work.

```
// Hello.java
class Hello {
  int test() {
    String s = null;
    return s.length();
  }
}
```

To run Infer, type the following in your terminal from the same directory as `Hello.java`.

```
infer run -- javac Hello.java
```

You should see the following error reported by Infer.

```
Hello.java:5: error: NULL_DEREFERENCE
  object s last assigned on line 4 could be null and is dereferenced at line 5
```

Now edit the file to add null checks:

```
  int test() {
    String s = null;
    return s == null ? 0 : s.length();
  }
```

Run Infer again. This time we get no error: Infer reports `No issues found`.

Hello world Objective-C

Here is a simple Objective-C example to illustrate Infer at work.

```
// Hello.m
#import <Foundation/Foundation.h>

@interface Hello: NSObject
@property NSString* s;
@end

@implementation Hello
NSString* m() {
    Hello* hello = nil;
    return hello->_s;
}
@end
```

To run Infer, type the following in your terminal from the same directory as `Hello.m`.

```
infer run -- clang -c Hello.m
```

You should see the following error reported by Infer.

```
Hello.m:10 NULL_DEREFERENCE
  pointer hello last assigned on line 9 could be null and is dereferenced at
line 10, column 12
```

Now edit the file to use the getter instead of accessing the instance variable:

```
NSString* m() {
    Hello* hello = nil;
    return hello.s;
}
```

Run Infer again. This time we get no error: Infer reports `No issues found`.

Hello world C

Here is a simple C example to illustrate Infer at work.

```
// hello.c
#include <stdlib.h>

void test() {
  int *s = NULL;
  *s = 42;
}
```

To run Infer, type the following in your terminal from the same directory as `hello.c`.

```
infer run -- gcc -c hello.c
```

You should see the following error reported by Infer.

```
hello.c:5: error: NULL_DEREFERENCE
  pointer s last assigned on line 4 could be null and is dereferenced at line 5,
column 10
```

Now edit the file to add null checks:

```
void test() {
  int *s = NULL;
```

```
  if (s != NULL) {
    *s = 42;
  }
}
```

Run Infer again. This time we get no error: Infer reports `No issues found`.

When analyzing C files, Infer captures the gcc command and runs clang instead to parse them. Thus you may get compiler errors and warnings that differ from gcc's. So in particular, the following two commands are equivalent:

```
infer run -- gcc -c hello.c
infer run -- clang -c hello.c
```

Hello world Android

To be able to analyze the sample Android app, make sure that you have the Android SDK 22 installed and up to date, as well as the "Android SDK Build-tools" and "Android Support Repository" components.

Go to the sample Android app in `infer/examples/android_hello` and create a `local.properties` file with a single line `sdk.dir=<location of your Android SDK>`. This sample Android app uses gradle as its build system. You do not need to install gradle to try it out though, thanks to the `gradlew` script in the project that will download gradle and the project's dependencies for you.

After editing `local.properties`, run

```
infer run -- ./gradlew build
```

Infer will output the list of found bugs:

```
app/src/main/java/infer/inferandroidexample/MainActivity.java:29: error:
NULL_DEREFERENCE
  object s last assigned on line 28 could be null and is dereferenced at line 29
  27.           setContentView(R.layout.activity_main);
```

```
  28.           String s = getDay();
  29. >         int length = s.length();
  30.           writeToFile();
  31.       }
  32.

app/src/main/java/infer/inferandroidexample/MainActivity.java:46: error:
RESOURCE_LEAK
   resource of type java.io.FileOutputStream acquired to fis by call to
FileOutputStream(...) at line 43 is not released after line 46
  44.               fis.write(arr);
  45.               fis.close();
  46. >         } catch (IOException e) {
  47.               //Deal with exception
  48.           }
  49.

app/src/main/java/infer/other/MainActivity.java:23: error: NULL_DEREFERENCE
  object returned by source() could be null and is dereferenced at line 23
  21.     @Override
  22.     protected void onCreate(Bundle savedInstanceState) {
  23. >     source().toString();
  24.     }
  25.
```

Differential analysis

If you run Infer again without changing any files, you will notice that this time nothing gets analyzed. This is because gradle is *incremental*: everything was compiled already so nothing gets recompiled. Infer captures the compilation commands to know which files to analyze, hence it analyzes nothing in this case. There are three solutions to remedy this:

1. Run gradlew clean in between Infer runs.

   ```
   ./gradlew clean
   ```

   This causes gradle to recompile everything each time, and subsequently Infer to capture all the files again.

2. Run Infer indicating that the capture of compilation commands should continue, using option `--continue` (or `-c` for short).

```
infer run --continue -- ./gradlew build
```

This makes Infer add the effects of the new compilation commands to the previous ones, and start a new analysis of the entire code.

3. Run Infer in reactive mode after a code change, using option `--reactive` (or `-r` for short).

```
infer run --reactive -- ./gradlew build
```

This makes Infer analyze the effects of the code change, without re-analyzing everything. Note that only the modified files, and those dependent on them, are re-analyzed. This analysis mode can be significantly faster.

You can learn more about the particulars of each solution in the Infer workflow page.

Hello world iOS

Go to the sample iOS app in `infer/examples/ios_hello` and run Infer on it:

```
infer run -- xcodebuild -target HelloWorldApp -configuration Debug -sdk
iphonesimulator
```

Infer will output the list of found bugs:

```
AppDelegate.m:20: error: MEMORY_LEAK
   memory dynamically allocated to shadowPath by call to CGPathCreateWithRect()
at line 20, column 28 is not reachable after line 20, column 5

AppDelegate.m:25: error: RESOURCE_LEAK
   resource acquired to fp by call to fopen() at line 25, column 8 is not
released after line 25, column 5

AppDelegate.m:29: warning: PARAMETER_NOT_NULL_CHECKED
```

```
   Parameter callback is not checked for null, there could be a null pointer
dereference: pointer callback could be null and is dereferenced at line 29,
column 5

AppDelegate.m:34: error: NULL_DEREFERENCE
   pointer str last assigned on line 33 could be null and is dereferenced at
line 34, column 12

AppDelegate.m:39: error: PREMATURE_NIL_TERMINATION_ARGUMENT
   pointer str last assigned on line 38 could be nil which results in a call to
arrayWithObjects: with 1 arguments instead of 3 (nil indicates that the last
argument of this variadic method has been reached) at line 39, column 12

Hello.m:20: error: NULL_DEREFERENCE
   pointer hello last assigned on line 19 could be null and is dereferenced at
line 20, column 12

Hello.m:25: warning: IVAR_NOT_NULL_CHECKED
   Instance variable hello -> _hello is not checked for null, there could be a
null pointer dereference: pointer ret_hello last assigned on line 24 could be
null and is dereferenced at line 25, column 12

Hello.m:30: warning: PARAMETER_NOT_NULL_CHECKED
   Parameter hello is not checked for null, there could be a null pointer
dereference: pointer ret_hello last assigned on line 29 could be null and is
dereferenced at line 30, column 12
```

Similarly to the case of gradle, running the command above a second time will yield no analysis results, as nothing gets recompiled. Either add the `--reactive` (or `-r`) flag to the `infer` command:

```
infer run --reactive -- xcodebuild -target HelloWorldApp -configuration Debug -
sdk iphonesimulator
```

or ask the build system to reinitialize the directory before running Infer again, using

```
xcodebuild -target HelloWorldApp -configuration Debug -sdk iphonesimulator clean
```

Hello world Make

Go to the sample C project in `infer/examples/c_hello` and run Infer on it:

```
infer run -- make
```

Infer will output the list of found bugs:

```
example.c:22: error: NULL_DEREFERENCE
   pointer max last assigned on line 21 could be null and is dereferenced at
line 22, column 10

example.c:36: error: NULL_DEREFERENCE
   pointer joe last assigned on line 35 could be null and is dereferenced by
call to get_age() at line 36, column 10

example.c:45: error: RESOURCE_LEAK
   resource acquired to fd by call to open() at line 41, column 12 is not
released after line 45, column 5

example.c:51: error: MEMORY_LEAK
   memory dynamically allocated to p by call to malloc() at line 51, column 14
is not reachable after line 51, column 3

example.c:57: error: MEMORY_LEAK
   memory dynamically allocated to p by call to malloc() at line 56, column 14
is not reachable after line 57, column 3
```

Similarly to the case of gradle, running `infer run -- make` a second time will yield no analysis results, as nothing gets recompiled. Either add the `--reactive` (or `-r`) flag to the `infer` command:

```
infer run --reactive -- make
```

or run

```
make clean
```