**Task 2: Escape Detection**

|  |  |  |
| --- | --- | --- |
| **Chew Yung Chung**  a0133662J  a0133662@u.nus.edu | **Tan Qiu Hao, Joel**  a0125473H  a0125473@u.nus.edu | **Juliana Seng**  a0126332R  a0126332@u.nus.edu |

Implementation and Algorithm

Within each function, the algorithm maintains a list of locally initialized pointers. When a return instruction is encountered, we check if the return operand matches any entry in the list of newly allocated pointers. If a match is found, an antipattern is detected.

For each function which is not the main() function, a check is performed. The code is as follows:

|  |  |
| --- | --- |
| 38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | StringMap<StringRef> allocaMap; //cout << "Function name:" << (&F)->getName().str() << endl; for (auto &BB: F) { // For each basic block BB  for (auto &I: BB) { // For each instruction I  if (I.getOpcode() == Instruction::Alloca) {  AllocaInst \*allocInstr = dyn\_cast<AllocaInst>(&I);  allocaMap[allocInstr->getName()] = "";  //cout << "New pointer name:" << allocInstr->getName().str() << "\n";  }  if (I.getOpcode() == Instruction::Ret) {  ReturnInst \*R = dyn\_cast<ReturnInst>(&I);  if (!R) continue;  Value \*v = R->getReturnValue();  if (!v) continue;  GlobalAlias \*ga = cast<GlobalAlias>(v);  if (!ga) continue;  Constant \*c = ga->getAliasee();  if (!c) continue;  c = cast<Constant>(c->stripPointerCasts());  //errs() << c->getName() << "\n";  //ReturnInst \*returnInstr = dyn\_cast<ReturnInst>(&I); if (R->getReturnValue() != nullptr) {  if (allocaMap.find(c->getName()) != allocaMap.end()) {  errs() << "WARNING: pointer <" << c->getName() << "> in the function <" << (&F)->getName().str() << "> will not exist after the return" << "\n";  }  }  }  }  } |

*Code snippet from asg2-task2.cpp*

For each instruction in the function, if the instruction is an LLVM Alloca instruction, the variable name is added to the **allocaMap**. Alloca instruction is performed when new pointers are created.

If the instruction is a return instruction, a check is performed to ensure that the return value is null, in which case, the escape violation will not occur and the program can proceed to check the next instruction. Otherwise, if the return value is not null, the variable name that is associated with the return value is retrieved and compared against the entries in **allocaMap**. If a match is found, the program concludes that the return value is a local pointer, and reports an escape violation.

Build and Run

Run build.sh to compile the test cases, program and execute the test cases.

|  |
| --- |
| $ ./build.sh |

The individual commands can also be found in build.sh.

An expected output of the shell code is as follows, all warnings are truncated.

|  |
| --- |
| ----------COMPILING TESTCASES----------  ----------COMPILING PROGRAM----------  ----------RUNNING TEST 1----------  WARNING: pointer <p> in the function <init\_array> will not exist after the return  ----------RUNNING TEST 2----------  WARNING: pointer <p> in the function <escape\_local> will not exist after the return |

Test Cases

Test case 1: DCL30-C Antipattern 1

|  |
| --- |
| $ ./asg2-task2 escape.ll |

Test case 1 observes the behaviour of escape.c, whereby the return value is a pointer to a locally created array.

|  |  |
| --- | --- |
| 7  8  9  10  11 | char \*init\_array(void) {  char array[10];  char \*p = array;  return p;  } |

*Code snippet from escape.c*

Test case 2: DCL30-C Antipattern 2

|  |
| --- |
| $ ./asg2-task2 escape2.ll |

Test case 2 observes the behaviour of escape2.c, whereby the return value is a pointer to a locally created character.

|  |  |
| --- | --- |
| 7  8  9  10  11 | char \*escape\_local() {  char local\_char = 'a';  char \*p = &local\_char;  return p;  } |

*Code snippet from escape2.c*

Limitations

Pointers to global variable

For this assignment, the algorithm does not take into account the possibility of the return value being a pointer pointing to a global variable. An example is as follows

|  |
| --- |
| int globalArr[10];  int\* escape\_func();  int main () {  return 0;  }  int\* escape\_func() {  int a[10];  int \*p = &globalArr;  return p;  } |

In the function escape\_func(), the pointer p points to a global array, hence an escape violation does not happen, and the statement is valid. However, our algorithm will raise a false positive as the return value p matches the newly allocated pointer p.