SE 421: Assignment #6

Due on October 19, 2018 at 12:00 PM (noon)

Instructor: Ben Holland

https://github.com/SE421/assignment6

Student Name:

Problem 1

(10 points)

Review the lecture notes and readings [A1, A2, A3] and then in your own words define the edge relationships of the following graphs. If you reference other sources in your literature review be sure to include appropriate citations in your report.

A. (*2 points*) Data Dependence Graph (DDG)

B. (*2 points*) Control Dependence Graph (CDG)

C. (*2 points*) Program Dependence Graph (PDG)

D. (*2 points*) System Dependence Graph (SDG)

(*2 points*) Of the four graphs listed above (DDG, CDG, PDG, SDG), which is the most difficult to construct and why?

References

[A1] M. Weiser, "Program slicing," in *Proceedings of the 5th international conference on Software engineering.* IEEE Press, 1981, pp. 439–449.

[A2] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 9, no. 3, pp. 319–349, 1987.

[A3] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 1, pp. 26–60, 1990.

Problem 2

(10 points)

In this problem, you are required to modify the provided skeleton code (`DataDependenceGraph.java`) to construct a data dependence graph (DDG) and the provided skeleton code (`ControlDependenceGraph.java`) to construct a control dependence graph (CDG). The union of the DDG and CDG results in the program dependence graph (PDG) that can be used to calculate the impact of changes to statements in software. Complete the tasks described by the commented "TODO"s.

Atlas smart views for the DDG, CDG and PDG have been provided to you as part of the skeleton code to help you debug your implementation. Additionally, an implementation of the Lengauer-Tarjan algorithm [B1] for computing the dominance analysis of the control flow graph has been provided to you in `DominanceAnalysis.java` that computes dominator trees and dominance frontiers. The dominance analysis is called automatically by the `PDGCodemapStage.java` class which invokes the dominance analysis computation before invoking your implementations of the DDG and CDG, so you can assume the new dominance analysis edges added by `DominanceAnalysis.java` will exist in the Atlas program graph before your code is invoked.

The provided code is an Eclipse plugin that depends on Atlas (`http://www.ensoftcorp.com/atlas`). You will be using the Atlas SDK (`https://www.ensoftcorp.com/atlas/sdk`) and its XCSG graph schema (`http://ensoftatlas.com/wiki/Extensible_Common_Software_Graph`) to extend the functionality of the provided plugin project. You may wish to use tools or utilities from your previous assignments.

A. (*5 points*) Data Dependence Graph (DDG) Construction
Commit your changes to the code on your GitHub assignment repository and briefly explain your modifications. Remember to provide a direct link to your source code on GitHub in your report!

B. (*5 points*) Control Dependence Graph (CDG) Construction
Again, commit your changes to the code on your GitHub assignment repository and briefly explain your modifications. Remember to provide a direct link to your source code on GitHub in your report!

References

[B1] T. Lengauer and R. E. Tarjan, "A fast algorithm for finding dominators in a flowgraph," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 1, no. 1, pp. 121–141, 1979.

Problem 3

Extra Credit (10 points)

In this problem you will use your implementation from Problem 2 to repeat the experiment shown in lecture to discover the buffer overflow vulnerability of the MiniShare 1.4.1 web server application [C1].

First import the provided `minishare` project then load the provided Atlas codemap. MiniShare 1.4.1 requires a particular build setup to properly map the project, so don't try to map the project yourself. Note that the provided Atlas codemap for `minishare` was produced with Atlas 3.3.7. If you are running an older version of Atlas you may need to check for updates (`Eclipse` → `Help` → `Check for Updates`). The codemap also already contains the "`control-dependence`" and "`data-dependence`" edges generated using the answer key.

Use the Atlas shell queries shown in Listing 1 below to locate the buffer overflow vulnerability.

Listing 1: Atlas Shell Queries

```
// get callers of strcpy and strlen
var strcpy = functions("strcpy")
var strlen = functions("strlen")
var callEdges = edges(XCSG.Call)
var strcpyCallers = callEdges.predecessors(strcpy)
var strlenCallers = callEdges.predecessors(strlen)

// select all array variables (variables have a TypeOf edge from an ArrayType)
var arrayTypes = nodes(XCSG.ArrayType)
var typeOfEdges = edges(XCSG.TypeOf)
var arrays = typeOfEdges.predecessors(arrayTypes)

// select structures that contain arrays
var arrayStructTypes = arrays.containers().nodes(XCSG.C.Struct)
var typeDefEdges = edges(XCSG.AliasedType, XCSG.TypeOf)
var typeAliases = nodes(XCSG.TypeAlias)
var arrayStructs = typeDefEdges.reverse(arrayStructTypes).difference(arrayStructTypes,
    typeAliases)

// buffers are arrays and structures containing arrays
var buffers = arrays.union(arrayStructs)

// find buffers that are tainted by attacker controlled inputs (the network socket)
var sockets = nodes(XCSG.Field).selectNode(XCSG.name, "socket")
var taint = universe.edges("control-dependence", "data-dependence")
var bufferStatements = buffers.containers().nodes(XCSG.ControlFlow_Node)
var taintedBufferStatements = taint.forward(sockets).intersection(bufferStatements)

// find strcpy callsites that take tainted buffers
var invocationEdges = edges(XCSG.InvokedFunction)
var strcpyCallsites = invocationEdges.predecessors(strcpy)
var strcpyCallsiteStatements = strcpyCallsites.containers().nodes(XCSG.ControlFlow_Node)
var taintedStrcpyCallsites = taintedBufferStatements.intersection(strcpyCallsiteStatements)

// find functions with tainted strcpy callsites that do not call strlen
var taintedStrcpyCallsiteFuncs = taintedStrcpyCallsites.containers().nodes(XCSG.Function)
var potentiallyVulnerableFuncs = taintedStrcpyCallsiteFuncs.difference(strlenCallers)

// filter tainted strcpy callsites to just callsites in the potentially vulnerable function
var taintedStrcpyCallsitesInPotentiallyVulnerableFuncs = taintedStrcpyCallsites.intersection
    (potentiallyVulnerableFuncs.contained().nodes(XCSG.ControlFlow_Node))
show(taintedStrcpyCallsitesInPotentiallyVulnerableFuncs)
```

Problem 3 continued on next page. . .

The Atlas queries in Listing 1 take some liberties. Think critically and answer the questions below with respect to the analysis performed by the queries in Listing 1 and your implementation of the PDG.

A. (*4 points*) The analysis provided does not attempt to find all possible buffer overflows but instead is looking for a particular class of buffer overflow vulnerabilities in the MiniShare application. In your own words, describe the high-level strategy this analysis uses to find candidates for buffer overflow vulnerabilities.

B. (*2 points*) If this analysis misses a buffer overflow that could be exploited by an attacker then we call it a false negative result. Provide an example of a false negative of this analysis (a buffer overflow that this analysis could miss)? You can suggest a hypothetical example (the example does not need to come from MiniShare).

C. (*2 points*) If this analysis identifies a buffer overflow that could not be exploited by an attacker then we call it a false positive result. Provide an example of a false positive of this analysis (a buffer overflow that this analysis could incorrectly identify as vulnerable)? You can suggest a hypothetical example (the example does not need to come from MiniShare).

D. (*2 points*) Does this analysis reveal a true positive result on MiniShare? If so what is the true positive result? How do you know the true positive result is a true positive?

References

[C1] "CVE-2004-2271." Available from MITRE, CVE-ID CVE-2004-2271., Jul. 17 2005. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2271