CS510 Assignment #2 (Due March 7th in class)

March 7, 2017

1 Dynamic Control Dependence (20p)

```
1.    if (p1)
2.       return;
3.    while (p2) {
4.       if (p3)
5.          break;
6.       if (p4) {
7.          s1;
8.       } else {
9.          s2;
10.         continue;
11.      }
12.      s3;
13.   }
14.   if (p5 ||
15.       p6) {
16.          s4;
17.   }
```

Consider the above code snippet. Assume the execution trace is 1, 3, 4, 6, 7, 12, 3, 4, 6, 9, 10, 3, 4, 5, 14, 16, 17.

Construct the dynamic control dependence subgraph, i.e., the graph that reveals control dependences between executed statements. Please show step-wise control dependence stack state and the dependence detected.

trace	cds	dep detected
1	(1_1, EXIT)	
3	(1_1, EXIT) (3_1, 14)	$3_1 \rightarrow 1_1$
4	(1_1, EXIT) (3_1, 14) (4_1, 14)	$4_1 \rightarrow 3_1$
6	(1_1, EXIT) (3_1, 14) (4_1, 14) (6_1, 3)	$6_1 \rightarrow 4_1$
7	(1_1, EXIT) (3_1, 14) (4_1, 14) (6_1, 3)	$7_1 \rightarrow 6_1$
12	(1_1, EXIT) (3_1, 14) (4_1, 14) (6_1, 3)	$12_1 \rightarrow 6_1$
3	(1_1, EXIT) (3_1, 14) (4_1, 14) (3_2, 14)	$3_2 \rightarrow 4_1$
4	(1_1, EXIT) (3_1, 14) (4_1, 14) (3_2, 14) (4_2, 14)	$4_2 \rightarrow 3_2$
6	(1_1, EXIT) (3_1, 14) (4_1, 14) (3_2, 14) (4_2, 14) (6_2,3)	$6_2 \rightarrow 4_2$
9	(1_1, EXIT) (3_1, 14) (4_1, 14) (3_2, 14) (4_2, 14) (6_2,3)	$9_1 \rightarrow 6_2$
10	(1_1, EXIT) (3_1, 14) (4_1, 14) (3_2, 14) (4_2, 14) (6_2,3)	$10_1 \rightarrow 6_2$
3	(1_1, EXIT) (3_1, 14) (4_1, 14) (3_2, 14) (4_2, 14) (3_3,14)	$3_3 \rightarrow 4_2$
4	(1_1, EXIT) (3_1, 14) (4_1, 14) (3_2, 14) (4_2, 14) (3_3,14) (4_3, 14)	$4_3 \rightarrow 3_3$
5	(1_1, EXIT) (3_1, 14) (4_1, 14) (3_2, 14) (4_2, 14) (3_3,14) (4_3, 14)	$5_1 \rightarrow 4_3$
14	(1_1, EXIT) (14_1, EXIT)	$14_1 \rightarrow 1_1$
16	(1_1, EXIT) (14_1, EXIT)	$16_1 \rightarrow 14_1$
17		

A number of students made the mistake of popping at the syntactic termination point of an if-statement. The right answer should be to pop at the immediate post-dominator. For example, popping an entry of statement 4 upon the execution of 6 is not right, that should happen upon the execution of 14.

2 Dynamic Data Dependence (20p)

```
1. void (* F) ();
2. char A[1];
3. char B[10];
4. int i,j;
5. i=j=0;
6. read(B, 10); //read 10 bytes
7. F= &foo();
8. while (j<10) {
9.   if (B[j]=='b')
10.     break;
11.  j=j++;
12.  if (j>0)
13.     i++;
14.  (*F) ();
15. }
16. A[i]=B[j];
17. (*F) ();
```

Data provenance tracking is a technique that tracks the set of INPUT VALUES that a variable or an executed statement is dependent on. For example, assume a program execution is

```
read (buf, 2) with input 10 and 20;
x=buf[0];
y=x+buf[1];
```

.

The provenance of x and y are $\{10\}$ and $\{10, 20\}$, respectively. Data provenance can be used to defend against code injection attacks by not allowing a function call to have a non-empty provenance.

(a) (10 points) Sketch a forward online algorithm that computes data provenance forwards along program execution, considering both data and control dependences.

(b) (10 points) Assume the input is "cb", apply your algorithm to the program at the beginning to detect code injection vulnerabilities. Note that function pointer F and array A are next to each other on the stack so that $A[1]$ shares the same memory location with the first byte of F.

Answer:
For each memory address x, we define a Pr(x) as the provence of the current value in x.

statement	action	comment
read(B, n)	for (i=0 to n) Pr(B+i)= id++	id stores t
y=op(x_1, x_2, ..., x_n)	Pr(&y)=Pr(&x_1)∪...∪Pr(&x_n)∪stack.top().third	
L:if (y)	stack.push(L, immediate_post_dom(L), Pr(&y)∪ stack.top().third	
L: an immediate post-dominator	while (stack.top().second=L) pop()	

The following shows the execution of the program. Symbol $\perp$ means not-input-related.

trace	Pr	stack
5. i=j=0;	Pr(&i)=Pr(&j)={}	
6. read(B, 10);	Pr(B)={0}; Pr(B+1)={1};	
7. F= &foo();	Pr(&F)={}	
8. while (j<10) {		[8, 16, {}]
9. if (B[j]=='b')		[8, 16, {}][9,16,{0}]
11. j=j++;	Pr(&j)={0}	
12. if (j>0)		[8, 16, {}][9,16,{0}][12,14,{0}]
13. i++;	Pr(&i)={0}	
14. (*F) ();		[8, 16, {}][9,16,{0}]
8. while (j<10) {		[8, 16, {}][9,16,{0}][8, 16, {0}]
9. if (B[j]=='b')		[8, 16, {}][9,16,{0}] [8, 16, {0}][9,16,{0,1}]
10 break		
16. A[i]=B[j];	Pr(A+1)=Pr(B+1)={1}	{ }
17. (*F) ();	warning as Pr(&F)={1}	

3 Formulating Dynamic Analysis (20p)

Please write the formal semantics for tracking dynamic control dependences on-the-fly. Please use the language that supports functions, which is the language on page 15 of the slides for program semantics. The output of the analysis shall be a trace of dynamic control dependences, which includes all the dynamic control dependences exercised during execution.

Language:

Program P ::= fd; s
Function f :: = M(y) { s }
FuncDef fd :: = f | fd; f
FuncId M, M1, M2, …
Statement s ::= s1; s2 | x=^{L} y | x =^{L} y op z | x=^{L} c |
if (x)^L s1 else s2 |
while (x)^L s | call(M, x)^L
Operation op :: = + | - | * | / | > | < | …
Value c :: = 0 | 1 | 2 … | true | false
Variable x, x1, x2, x3

(a) The semantics configuration is $< s, \delta, Cnt, Cds, Cxt, D > \rightarrow < s', \delta', Cnt, Cds', Cxt', D' >$
ControlDepStack Cds: (Label × Int × Context × Label)*
For example, <L1, 1, L2•L3, L4> as the top entry of the CDS means that the execution is currently in the region led by the first instance of L1, the region will end at L4 (the immediate post-dominator of L1) with the calling context of L2•L3

Counter Cnt: Label -> Int
DynControlDep D: P(Label × Int × Label × Int)

(b) Semantics rules

$$\frac{\begin{array}{c}\boldsymbol{\delta' = \delta[x \mapsto c] \quad Cnt' = Cnt[L \rightarrow Cnt[L] + 1]} \\ \boldsymbol{Cds = < Lp, i > \cdot T \quad D' = D \cup \{< L, Cnt[L], Lp, i >\}}\end{array}}{\boldsymbol{< x =^L c; s, \delta, Cnt, Cds, D > \rightarrow < s, \delta', Cnt', Cds, D' >}}$$

$$\frac{\begin{array}{c}\boldsymbol{Cnt' = Cnt[L \rightarrow Cnt[L] + 1] \quad Cds' = < L, cnt[L] > \cdot Cds} \\ \boldsymbol{Cds = < Lp, i > \cdot T \quad D' = D \cup \{< L, Cnt[L], Lp, i >\}} \\ \boldsymbol{< s1, \delta, Cnt', Cds', D' > \rightarrow < skip, \delta', Cnt'', Cds'', D'' > \quad \delta[x] == true}\end{array}}{\boldsymbol{< if\ (x)^L\ s1\ else\ s2; s, \delta, Cnt, Cds, D > \rightarrow < s, \delta', Cnt'', Cds, D'' >}}$$

$$\frac{\begin{array}{c}\boldsymbol{Cnt' = Cnt[L \rightarrow Cnt[L] + 1] \quad Cds' = < L, cnt[L] > \cdot Cds} \\ \boldsymbol{Cds = < Lp, i > \cdot T \quad D' = D \cup \{< L, Cnt[L], Lp, i >\} \quad M(y)\{s0\}} \\ \boldsymbol{< s0, \delta, Cnt', Cds', D' > \rightarrow < skip, \delta', Cnt'', Cds'', D'' >}\end{array}}{\boldsymbol{< M(x)^L; s, \delta, Cnt, Cds, D > \rightarrow < s, \delta', Cnt'', Cds, D'' >}}$$

4 Static Analysis (20p)

Design an analysis to determine the sign of the possible values of a variable, all negative numbers by the symbol -, zero by the symbol 0, and all positive numbers by the symbol +. Assume only int type is supported. Only two kinds of binary operations are possible: addition and subtraction. There may be predicates and loops.

(a)

$$< V, W, s, D, X > \rightarrow < V', W', s', D', X' >$$

- Sign: 0|+|-| *
- SignStore D: Variable -> Sign
- VarSign X: P (Variable × Sign)
- WorkList W: P (Definition × Statement)
- Visited V: P (Definition × Statement)

(b)

$$\frac{D' = D[x \longmapsto +] \quad c > 0 \quad X' = X \cup \{< x, + >\}}{< V, W, x =^L c; s, D, X > \rightarrow < V, W, s, D', X' >}$$

$$\frac{D' = D[x \longmapsto 0] \quad c = 0 \quad X' = X \cup \{< x, 0 >\}}{< V, W, x =^L c; s, D, X > \rightarrow < V, W, s, D', X' >}$$

$$\frac{D' = D[x \longmapsto -] \quad c < 0 \quad X' = X \cup \{< x, - >\}}{< V, W, x =^L c; s, D, X > \rightarrow < V, W, s, D', X' >}$$

$$\frac{D' = D[x \longmapsto +] \quad D[y] = + \quad D[z] = + \quad X' = X \cup \{< x, + >\}}{< V, W, x =^L y + z; s, D, X > \rightarrow < V, W, s, D', X' >}$$

$$\frac{D' = D[x \longmapsto +] \quad D[y] = + \quad D[z] = 0 \quad X' = X \cup \{< x, + >\}}{< V, W, x =^L y + z; s, D, X > \rightarrow < V, W, s, D', X' >}$$

$$\frac{D' = D[x \longmapsto +] \quad D[y] = 0 \quad D[z] = + \quad X' = X \cup \{< x, + >\}}{< V, W, x =^L y + z; s, D, X > \rightarrow < V, W, s, D', X' >}$$

$$\frac{D' = D[x \longmapsto 0] \quad D[y] = 0 \quad D[z] = 0 \quad X' = X \cup \{< x, 0 >\}}{< V, W, x =^L y + z; s, D, X > \rightarrow < V, W, s, D', X' >}$$

$$\frac{D' = D[x \longmapsto *] \quad D[y] = + \quad D[z] = - \quad X' = X \cup \{< x, * >\}}{< V, W, x =^L y + z; s, D, X > \rightarrow < V, W, s, D', X' >}$$

$$\frac{D' = D[x \longmapsto *] \quad D[y] = * \quad X' = X \cup \{< x, * >\}}{< V, W, x =^L y + z; s, D, X > \rightarrow < V, W, s, D', X' >}$$

$$\frac{D' = D[x \mapsto *] \quad D[z] = * \qquad X' = X \cup \{<x,*>\}}{<V,W,x =^L y + z; s, D, X> \rightarrow <V,W,s,D',X'>}$$

$$\frac{D' = D[x \mapsto *] \quad D[y] = - \quad D[z] = + \quad X' = X \cup \{<x,*>\}}{<V,W,x =^L y + z; s, D, X> \rightarrow <V,W,s,D',X'>}$$

$$\frac{D' = D[x \mapsto -] \quad D[y] = - \quad D[z] = - \quad X' = X \cup \{<x,->\}}{<V,W,x =^L y + z; s, D, X> \rightarrow <V,W,s,D',X'>}$$

$$\frac{D' = D[x \mapsto -] \quad D[y] = - \quad D[z] = 0 \quad X' = X \cup \{<x,+>\}}{<V,W,x =^L y + z; s, D, X> \rightarrow <V,W,s,D',X'>}$$

$$\frac{D' = D[x \mapsto -] \quad D[y] = 0 \quad D[z] = - \quad X' = X \cup \{<x,+>\}}{<V,W,x =^L y + z; s, D, X> \rightarrow <V,W,s,D',X'>}$$

The rules for subtraction are similar and hence omitted.

$$\frac{\begin{array}{c} W' = W \cup <D, s2; s> \quad \neg <D, s2; s> \ni V \\ \neg <D, s1; s> \ni V \\ V' = V \cup <D, s2; s> \cup <D, s1; s> \end{array}}{<V, W, if\ (x^L)\ s1\ else\ s2; s, D, X> \rightarrow <V',\ W', s1; s, D, X>}$$

$$\frac{}{\begin{array}{c} <V, W, while\ (x) s1; s, D, X> \rightarrow \\ <V, W, if\ (x)\ s1; while\ (x)\ s1\ else\ skip; s, D, X> \end{array}}$$

(c) V is monotonically growing and it has a finite domain.

(d) Configuration $<s, D, X> \rightarrow <s',\ D', X'>$

$$\frac{\begin{array}{c} <s1, D, X> \rightarrow <skip, D1, X1> \\ <s2, D, X> \rightarrow <skip, D2, X2> \\ \forall x, D'[x \rightarrow D1[x] \bowtie D2[x]] \qquad X' = X1 \cup X2 \end{array}}{<if\ (x^L)\ s1\ else\ s2; s, D, X> \rightarrow <s, D', X'>}$$

The operator ⋈ The operator is defined as + ⋈+=+ , + ⋈**0=+** , + ⋈**-=*,...**

$$\frac{\begin{array}{c} <s1, D, X> \rightarrow <skip, D1, X1> \\ \neg D1 \subseteq D \\ \forall x, D'[x \rightarrow D1[x] \bowtie D[x]] \qquad X' = X \cup X1 \end{array}}{\begin{array}{c} <while\ (x) s1; s, D, X> \rightarrow \\ <while\ (x)\ s1; s, D', X'> \end{array}}$$

The operator $\subseteq$ The operator defines a partial order, we say $\boldsymbol{D1} \subseteq \boldsymbol{D}$ iff D1[x] <= D[x] for all x, with nil <= 0,+,- and 0,+,- <= *

$$\frac{\begin{array}{c} < \boldsymbol{s1}, \boldsymbol{D}, \boldsymbol{X} > \to < \boldsymbol{skip}, \boldsymbol{D1}, \boldsymbol{X1} > \\ \boldsymbol{D1} \subseteq \boldsymbol{D} \end{array}}{\begin{array}{c} < \boldsymbol{while}\ (\boldsymbol{x})\boldsymbol{s1}; \boldsymbol{s}, \boldsymbol{D}, \boldsymbol{X} > \to \\ < \boldsymbol{s},\ \boldsymbol{D}, \boldsymbol{X1} > \end{array}}$$