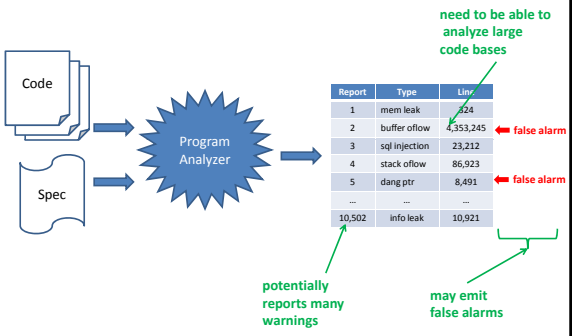
Program Analysis for Security

Suhabe Bugrara
Stanford University

Web Application Code

```
1. javax.sql.Connection con = . . .;

2. javax.servlet.http.HttpServletRequest request = . . .;

3. String username = request.getParameter("username");

4. String query = "SELECT * FROM Users " +
                  " WHERE name = '" + username + "'";

5. con.execute(query);
```

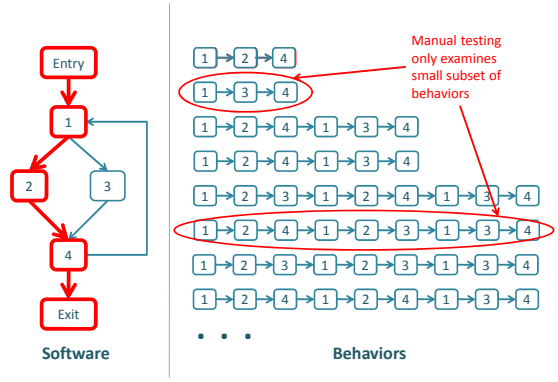
Program Analyzers

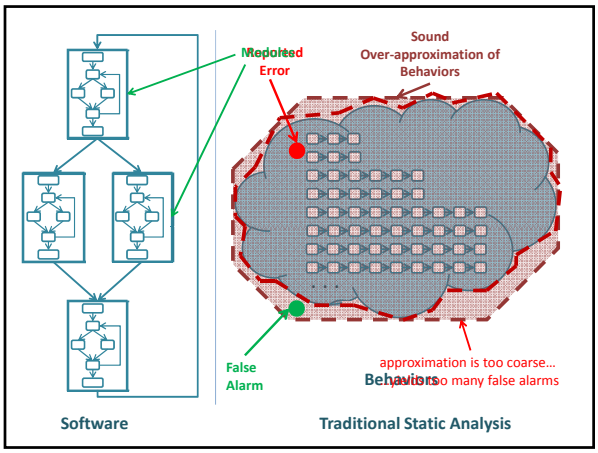


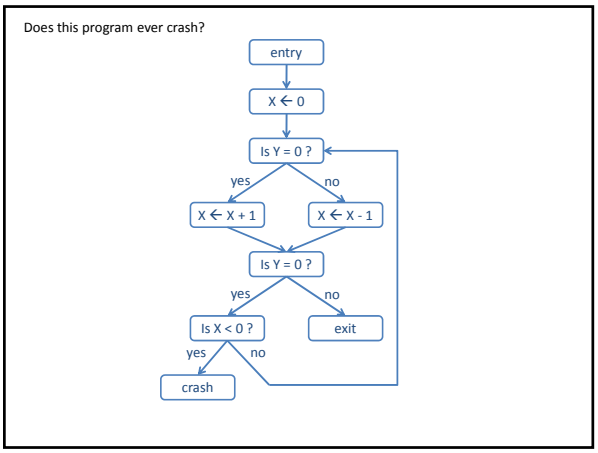
Soundness, Completeness

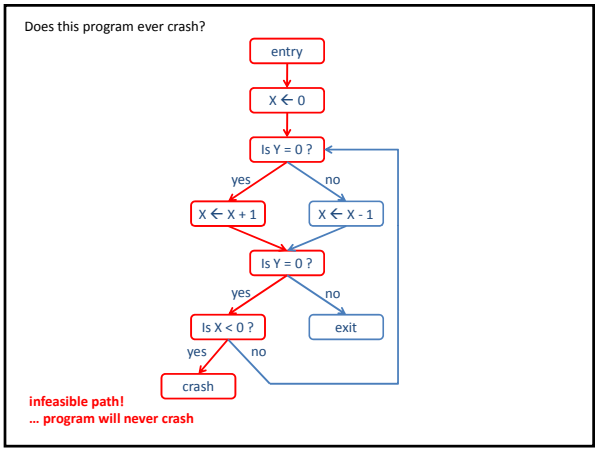
Dimension	Definition
Soundness	If the program contains an error, the analysis will report a warning.
Completeness	If the analysis reports an error, the program will contain an error.

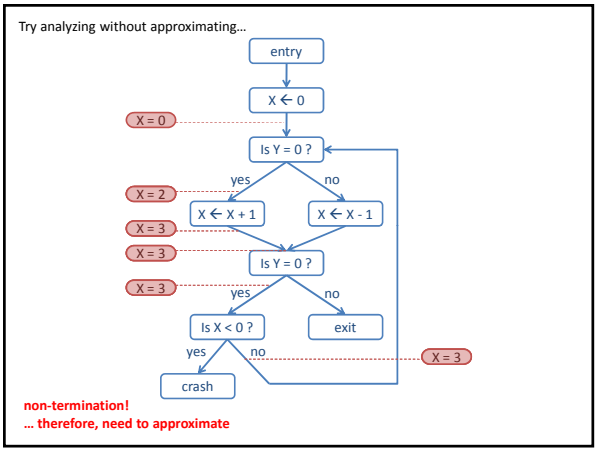
	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms **Decidable**
Unsound	May not report all errors Reports no false alarms **Decidable**	May not report all errors May report false alarms **Decidable**

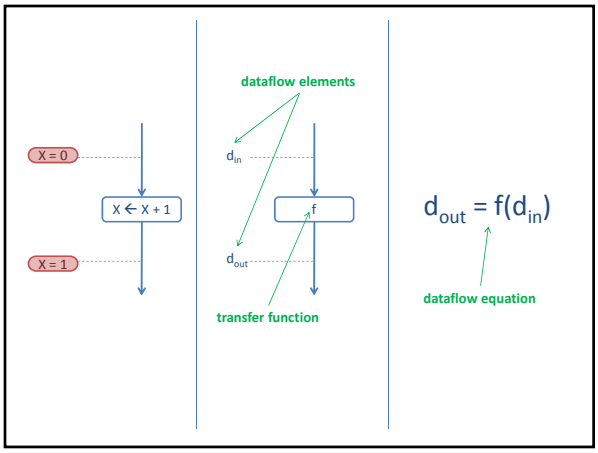


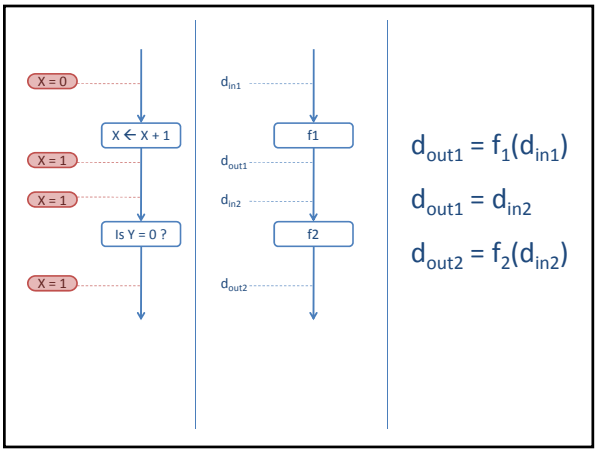

Reported
Error
Sound
Over-approximation of
Behaviors
False
Alarm
approximation is too coarse...
... too many false alarms
Behaviors
Software
Traditional Static Analysis

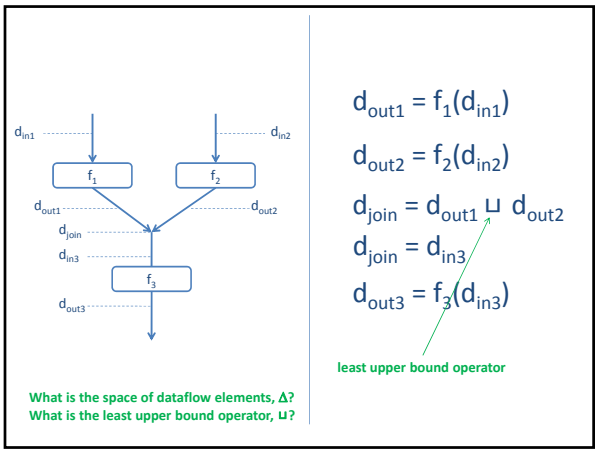

Does this program ever crash?
entry
X ← 0
Is Y = 0 ?
yes
no
X ← X + 1
X ← X - 1
Is Y = 0 ?
yes
no
Is X < 0 ?
exit
yes
no
crash

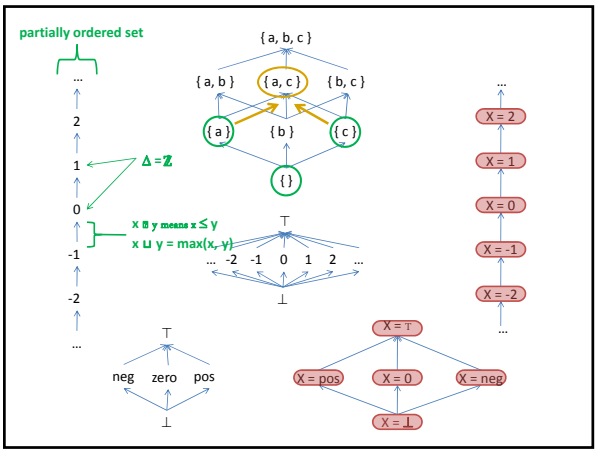

Does this program ever crash?
entry
X ← 0
Is Y = 0 ?
yes
no
X ← X + 1
X ← X - 1
Is Y = 0 ?
yes
no
Is X < 0 ?
exit
yes
no
crash
infeasible path!
... program will never crash

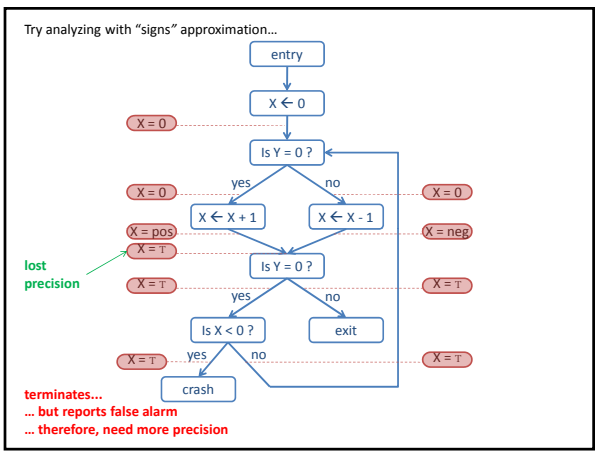

Try analyzing without approximating...
entry
X ← 0
X = 0
Is Y = 0 ?
yes
no
X = 2
X ← X + 1
X ← X - 1
X = 3
X = 3
Is Y = 0 ?
X = 3
yes
no
Is X < 0 ?
exit
yes
no
X = 3
crash
non-termination!
... therefore, need to approximate

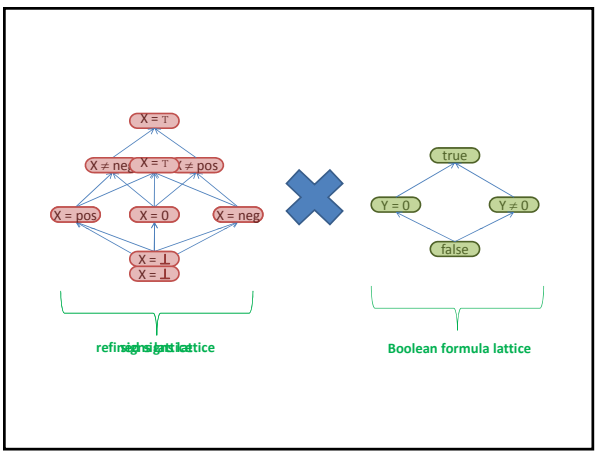

X = 0
X ← X + 1
X = 1
dataflow elements
d_{in}
f
d_{out}
transfer function
$d_{out} = f(d_{in})$
dataflow equation

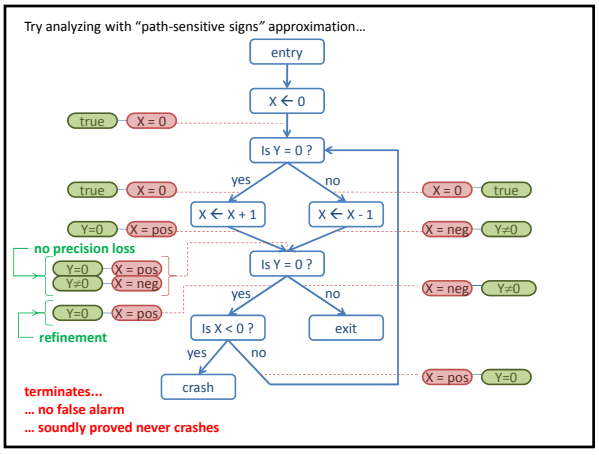

X = 0
X ← X + 1
X = 1
X = 1
Is Y = 0 ?
X = 1
d_{in1}
f1
d_{out1}
d_{in2}
f2
d_{out2}
$d_{out1} = f_1(d_{in1})$
$d_{out1} = d_{in2}$
$d_{out2} = f_2(d_{in2})$



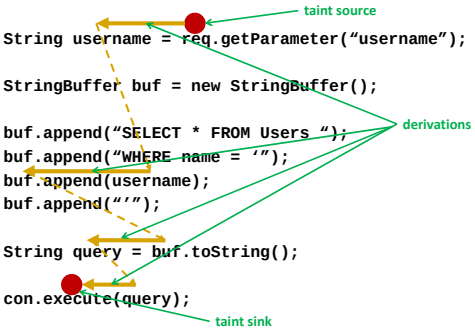








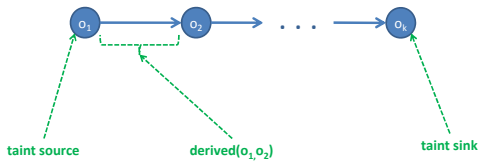
Tainted Object Propagation



Tainted Object Propagation

Term	Descriptor	Example
Source Object	Method	HttpServletRequest.getParameter(String)
	Parameter	return
	Access path	ε
Sink Object	Method	Connection.execute(String)
	Parameter	first argument
	Access path	ε
Derivation	Method	StringBuffer.append(String)
	From Parameter	first argument
	From Access Path	ε
	To Parameter	this
	To Access Path	ε

Security Violation



Complication of Aliasing

```
String username = req.getParameter("username");

StringBuffer buf1 = new StringBuffer();
StringBuffer buf2 = buf1;

buf1.append(username);

String query = buf2.toString();

con.execute(query);
```

$\exists o$. *points-to*(buf1,o) ∧ *points-to*(buf2,o)

analyzer must know about aliasing relationship between buf1 and buf2 to find vulnerability

No security violation found!

Statements

Statement	Code	Description
object creation	o_i: `T v = new T();`	Creates a new heap object o_i of type **T**, and makes variable **v** point to o_i
copy	`v = w;`	Makes **v** point to whatever heap object **w** currently points to
field store	`v.f = w;`	Let **v** point to heap object o_1. Let **w** point to heap object o_2. Makes the field **f** in o_1 now point to o_2.
field load	`v = w.f;`	Let **w** point to heap object o_1. Makes **v** point to whatever the field **f** in o_1 points to.

Pointer Analysis

Predicate	Description
points-to (**v**,o)	variable **v** can point to heap object o
heap-points-to (o_1,**f**,o_2)	field **f** of heap object o_1 can point to heap object o_2

Predicate	Rule
points-to (**v**, o_1)	"o_1: `T v = new T();`"
points-to (**v**, o_1)	"`v = w;`"
	points-to (**w**, o_1)
heap-points-to (o_1,**f**,o_2)	"`v.f = w;`"
	points-to (**v**, o_1)
	points-to (**w**, o_2)
points-to (**v**, o_2)	"`v = w.f;`"
	points-to (**w**, o_2)
	heap-points-to (o_1,**f**,o_2)

Statement	sql-injection (o_{src}, o_{sink})
`1: user = req.getParam("user");`	ret (i_1, v_1)
	call (i_1,"`HttpServletRequest.getParameter(String)`")
	points-to (v_1, o_{src})
`2: buf.append(user);`	actual (i_2, v_2, 0)
	actual (i_2, v_3, 1)
	call (i_2, "`StringBuffer.append(String)`")
	points-to (v_2, o_{temp})
	points-to (v_3, o_{src})
`3: query = buf.toString();`	actual (i_3, v_4, 0)
	ret (i_3, v_5)
	call (i_3, "`StringBuffer.toString(String)`")
	points-to (v_4, o_{temp})
	points-to (v_5, o_{sink})
`4: con.execute(query);`	actual (i_4, v_6, 1)
	call (i_4, "`Connection.execute(String)`")
	points-to (v_6, o_{sink})

Context Sensitivity

```
String passedUrl = request.getParameter("...");
DataSource ds1 = new DataSource(passedUrl);

String localUrl = "http://localhost/";
DataSource ds2 = new DataSource(localUrl);

String s1 = ds1.getUrl();
String s2 = ds2.getUrl();

StringBuffer buf1 = new StringBuffer();
buf1.append(s2);

String query = buf1.toString();

Connection con = ...;

con.execute(query);  false alarm!
```

```
class DataSource {
    private String url;

    public DataSource(String url) {
          this.url = url;
    }

    String getUrl() {
                return this.url;
    }
}
```

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms **Decidable**
Unsound	May not report all errors Reports no false alarms **Decidable**	May not report all errors May report false alarms **Decidable**

```
int bad_abs (int x) {
        if (x < 0)
                return –x;

        if (x == 12345678)
                return –x;

        return x;
}
```

Constraint
(x >= INT_MIN) && (x <= INT_MAX) && (x < 0) && (ret = -x)

```
int bad_abs (int x) {
        if (x < 0)
                return –x;

        if (x == 12345678)
                return –x;

        return x;
}
```

Constraint
(x >= INT_MIN) && (x <= INT_MAX) && (x >= 0) && (x = 12345678) && (ret = -x)

Solution
x = 12345678

```
int bad_abs (int x) {
        if (x < 0)
                return –x;

        if (x == 12345678)
                return –x;

        return x;
}
```

Constraint
(x >= INT_MIN) && (x <= INT_MAX) && (x >= 0) && (x != 12345678) && (ret = x)

Solution
x = 4

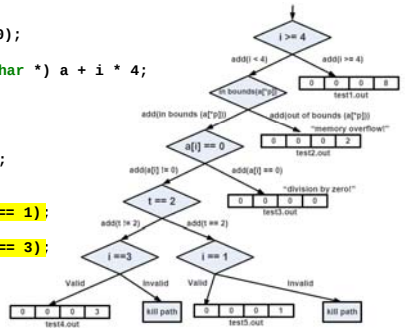
```
int bad_abs (int x) {
        if (x < 0)
                return –x;

        if (x == 12345678)
                return –x;

        return x;
}
```

KLEE automatically generated test cases for each path…

x = -1
x = 12345678
x = 4

```
1: int symbolic_bad_abs (int x) {
2:     add_constraints(x >= INT_MIN, x <= INT_MAX);
3:     ret = new symbol;
4:
5:     if (fork() == child) {
6:             add_constraints(x < 0, ret = -x);
7:             return ret;
8:             //(x >= INT_MIN) && (x <= INT_MAX) && (x < 0) && (ret = -x)
9:     } else
10:            add_constraints(x >= 0);
11:
12:    if (fork() == child) {
13:            add_constraints(x = 12345678, ret = -x);
14:            return ret;
15:            //(x >= INT_MIN) && (x <= INT_MAX) && (x >= 0) && (x = 12345678)
16:            //      && (ret = -x)
17:    } else
18:            add_constraints(x != 12345678);
19:
20:    add_constraints(ret = x);
21:    return ret;
22:    //(x >= INT_MIN) && (x <= INT_MAX) && (x >= 0) && (x != 12345678)
23:            && (ret = x)
24:}
```

```
1: int main (void) {
2:     unsigned i, t, a[4] = { 1, 3, 5, 2};
3:     make_symbolic(&i);
4:
5:     if (i >= 4)
6:             exit(0);
7:
8:     char *p = (char *) a + i * 4;
9:     *p = *p - 1;
10:
11:    t = a[*p];
12:
13:    t = t / a[i];
14:
15:    if (t == 2)
16:      assert (i == 1);
17:    else
18:      assert (i == 3);
19: }
```



Questions