Wrap-Up

December 5, 2011
What do I think you know?

- Exam
- Sharir-Pnueli
- Key elements of papers
Exam

• Vulnerability – definition

• Static analysis – Chapter 2
  ‣ Definitions for basic concepts
  ‣ Abstract domain
  ‣ Dataflow problem
  ‣ Join/Meet and Path
  ‣ Join-over-all-paths
  ‣ Join-over-all-valid-paths
Exam

• Somewhat more complex topics
  ‣ Flow-sensitivity or not … Context-sensitivity or not
    • Would you know it if you saw it applied?
  ‣ Configurations and past/future
    • P-automaton and prestar and poststar and
    • What’s the point? Relate concepts to specifics?
  ‣ Weighted pushdown system concepts
    • Assume you know what an FSA and PDA are
    • Good for asking about dataflow problems
    • Probably I’ll have to explain a bit
Exam

• Lots of abstract domains and composition functions and join/meet functions
  ‣ What do they mean? How might they affect results?
  ‣ I’ll have to help with these
Exam

• Key concepts
  ‣ Every paper has a key concept
  ‣ What do you think it is?
Return-oriented programming

• The execution model
• Instruction pointer is stack
  ‣ Followed by executing until a return occurs
  ‣ Data is also on the stack (push and pop into registers)
  ‣ Connecting gadgets together
Control-Flow Integrity

• Approach enforces possible valid control flows (paths)
  ‣ A calls B at instruction X
  ‣ B must return to X+1

• However, there are difficulties due to imprecision
  ‣ What are these and how are they dealt with?
Metal and MC

- Cast bug finding as a dataflow problem
- Each variable is associated with a state
- Transition rules change among states
  - Source state, pattern, destination state
- Dataflow problem
  - ICFG
  - Join semilattice
  - Initial value
  - Assignment
Information Flow Analysis

• Systems and programs define data flows
  ‣ How do you make a graph?

• Information flow policy as lattice

• Some nodes are labeled using lattice levels

• Find information flow errors
  ‣ What is an information flow error?
  ‣ How does this relate to dataflow problem?
SAT Solvers

- Several different techniques applied
- The exam required Stalmark and Sakallah
- How do those work?
Compiler

• Ccured has a specific goal
  ‣ What is it?

• LLVM paper was about vision
  ‣ What is their vision?
Namespaces

• Each paper has a major claim
  ‣ What are they?
  ‣ What do they mean?

• Chari et al

• Cai et al
Attack Graphs

• MulVal
  ‣ How does it express attacks?

• Datalog
  ‣ Clauses
  ‣ Limitations

• Our approach
  ‣ Information flow and cuts
• Call Strings

Call String (CS) approach

main

read a, b
t := a * b

τ_{main}
c_1
call p

a*b avail.?
t := a * b
print t

n_2
stop

\( e_{main} \)

\( p \)

if a=0

T

τ_{p}

\[ \text{if } a=0 \]

\[ a := a - 1 \]

\[ \text{call } p \]

\[ t := a * b \]

\[ \text{return} \]

\[ \text{c}_2 \]

12.06.2010
Nikolai Knopp
• Call Strings

Call String (CS) approach

```
main

read a, b
  t := a * b

if a=0
  a := a - 1
  call p
else
  t := a * b
  print t

a*b avail.?
  return

stop

c1

rmain

c2

ep

rp

return

r
```

12.06.2010 Nikolai Knopp
Sharir-Pnueli

• Call Strings

Call String (CS) approach

- main
  - (⟨⟩, χ)
  - call p
  - a*b avail.? t := a * b
  - print t
  - stop
  - e_{main}

- p
  - if a=0
    - a := a - 1
    - call p
  - t := a * b
  - return
  - e_p

12.06.2010 Nikolai Knopp
• Call Strings

![Diagram showing Call String (CS) approach]

- `main`
  - read a, b
  - t := a * b
  - (⟨⟩, 1)
  - `τ_{main}`
  - `ε_{main}`

- `a*b avail.?`
  - t := a * b
  - print t
  - stop

- `p`
  - if a=0
    - `τ_{p}`
    - `ε_{p}`
  - a := a - 1
  - call p
  - t := a * b
  - return

12.06.2010
Nikolai Knopp
Sharir-Pnueli

• Call Strings

Call String (CS) approach

```
main
read a, b
t := a * b
```

```
p
((c_1), 1)
a := a - 1
```

```
c_1
```

```
T
```

```
c_2
```

```
F
```

```
r_{main}
```

```
e_{main}
```

```
r_p
```

```
e_p
```

```
t := a * b
```

```
print t
```

```
call p
```

```
call p
```

```
stop
```

```
return
```

12.06.2010
Nikolai Knopp
• Call Strings
• Call Strings

**Call String (CS) approach**

```
main
  read a, b
  t := a * b
  call p
  a*b avail.?
  t := a * b
  print t
  stop

  r_{main}
  c_1
  u_2
  e_{main}
```

```
p
  if a=0
    a := a - 1
    ((c_1), T)
  return

  r_p
  c_2
  e_p
```

12.06.2010
Nikolai Knopp
Call Strings

Call String (CS) approach

```
main
read a, b
  t := a * b
  call p
  a*b avail.?  
  t := a * b
  print t
  stop

\( p \)
  \((c_1c_2), T)\)
  a := a - 1
  call p
  \(T\)
  t := a * b
  return
```

12.06.2010
Nikolai Knopp
Sharir-Pnueli

• Call Strings
• Call Strings

Call String (CS) approach

- **main**
  - read a, b
  - \( t := a \times b \)
  - call p
  - \( \text{a*b avail.?} \)
  - \( t := a \times b \)
  - print t
  - stop

- **p**
  - if a=0
    - T
    - \( a := a - 1 \)
    - call p
  - F
  - (\( c_1, T \))
  - return

12.06.2010
Nikolai Knopp
Sharir-Pnueli

- Call Strings

Call String (CS) approach

```
main
read a, b  
t := a * b  
call p
read a, b  
t := a * b  
print t
stop
```

```
p
if a=0
    a := a - 1
    call p
    t := a * b
    (c₁, 1)
```

```
a * b avail.?
    t := a * b
```

12.06.2010 Nikolai Knopp
• Call Strings

Call String (CS) approach

```
main
  read a, b
  t := a * b
  call p
  a*b avail.? ((), 1)
  print c
  stop
  r_{main}
  e_{main}

p
  if a=0
    T
    a := a - 1
    call p
    c_2
    e_p
  F
  t := a * b
  return
  r_p
```

12.06.2010
Nikolai Knopp
Sharir-Pnueli

- Call Strings

- Let $q \in IVP(r_{main}, n)$ decomposed as:

- $\langle c_1 c_2 ... c_j \rangle =: \gamma \in \Gamma$

- call string (CS) to $q$ in $G^*$.

- $\lambda \in \Gamma$ is empty call string ($j = 0$ in $main$).

- $\Gamma = \text{space of valid call strings in } G^*$

- $CM : IVP \to \Gamma \text{ with } CM(q) = \gamma$
Sharir-Pnueli

- Tracks calls, returns
- Prevents invalid flows
  - Why?
Exam

• 13 - Context inlining
  ‣ Versus summary function approach

• 14 - GRASP (Sakallah approach)
  ‣ Adding new constraints

• 15 – Ccured
  ‣ Qualifiers? SAFE, SEQ, DYNQ
  ‣ Constraints on qualifier values – find valid solution (ARITH, CONV, POINTSTO, TYPEEQ)
  ‣ Constraint solving and minimality
Exam

• 16 - ROP
  ‣ Stack 10, 20, 30, 50
  ‣ Add constant: 10, CONST, 20, 30, 50
  ‣ Gadget 10 must push output, and gadget 20 must pop constant and output

• 17 – Creative

• 18 – abstract domain and dataflow problem
  ‣ Domain: A set of states defined by rules
  ‣ CFG: CFG
  ‣ Join – probably a union
  ‣ Initial value is null
  ‣ Assignments – transitions in rules
• 19 – Code
  ‣ (a) PDS: should be able to do that
  ‣ (b, c) Valid flow – should be able to identify valid and invalid flows
  ‣ (d) P-automaton
    • Configuration \{<p, e_{main}>\}
    • \(P \rightarrow e_{main} \rightarrow \text{accept}\)
    • Configuration \{<p, e_{main} \ldots n6n9>\}
    • \(P \rightarrow \text{sequence of transitions for valid} \rightarrow \text{accept}\)
(e) Prestar

- Configuration \( \{<p, e_{\text{main}} \ldots n6n9>\} \)
- \((p, n5) \rightarrow (p, n6) \land (p, n6) \rightarrow (p, n9) \land (p, n9) \rightarrow (acc, e)\)
- \((p, n5, acc)\)
- Basically, all reachability paths to n9 via configuration (i.e., in the P-Automaton) lead to accepting state
Exam

• 20 – Policy
  ‣ (a) Build dataflow graph
  ‣ (b) Reachability from and to t
  ‣ (c) Stoller rule-specific
  ‣ (d) Stoller TCB
  ‣ (e) DLM – intersection of readers