Lecture 17 – Dataflow Analysis (and more!)

THEORY OF COMPILATION

Eran Yahav

www.cs.technion.ac.il/~yahave/tocs2011/compilers-lec17.pptx

Reference: Dragon 9, 12

1

Last week: dataflow

- General recipe
  - design the domain
  - transfer functions
  - determine join operation (may/must?)
  - direction: forward/backward?
- note initial values

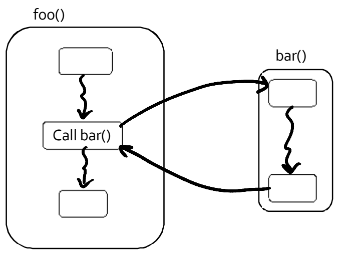
2

Analyses Summary

	Reaching Definitions	**Available Expressions**	**Live Variables**
L	$\wp$(Var x Lab)	$\wp$(AExp)	$\wp$(Var)
$\sqsubseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\sqcup$	$\cup$	$\cap$	$\cup$
$\perp$	$\emptyset$	AExp	$\emptyset$
Initial	$\{ (x,?) \mid x \in Var\}$	$\emptyset$	$\emptyset$
Entry labels	{ init }	{ init }	final
Direction	Forward	Forward	Backward
F	$\{ f: L \longrightarrow L \mid \exists k,g : f(val) = (val \setminus k) \cup g \}$		
f_{lab}	$f_{lab}(val) = (val \setminus kill) \cup gen$		

3

Interprocedural Analysis



- The effect of calling a procedure is the effect of executing its body

4

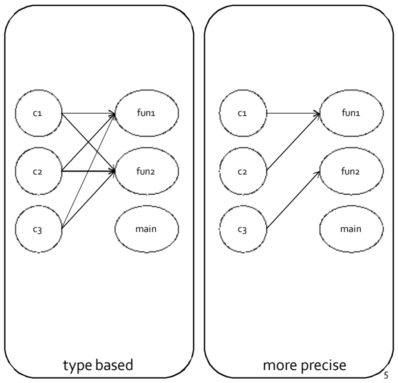
Call Graph

```
int (*pf)(int);

int fun1 (int x) {
  if (x < 10)
    return (*pf) (x+l); // C1
  else
    return x;
}

int fun2(int y) {
  pf = &fun1;
  return (*pf) (y) ; // C2
}

void main() {
  pf = &fun2;
  (*pf )(5) ; // C3
}
```

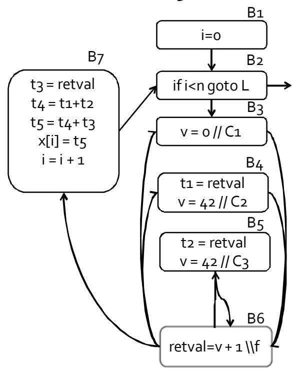


5

Context Sensitivity

```
main() {
 for (i = 0; i < n; i++) {
  t1 = f(0);   // C1
  t2 = f(42); // C2
  t3 = f(42); // C3
  X[i] = t1 + t2 + t3;
 }
}

int f(int v)
  return (v+1);
}
```



6

Solution Attempt #1

- Inline callees into callers
  - End up with one big procedure
  - CFGs of individual procedures = duplicated many times
- Good: it is precise
  - distinguishes different calls to the same function
- Bad
  - exponential blow-up, not efficient
  - doesn't work with recursion

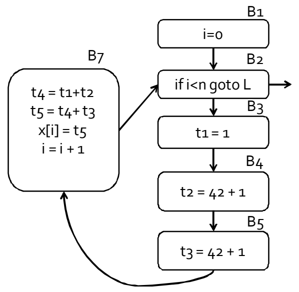
```
main() { f(); f(); }
f() { g(); g(); }
g() { h(); h(); }
h() { ... }
```

7

Inlining

```
main() {
 for (i = 0; i < n; i++) {
  t1 = f(0);   // C1
  t2 = f(42); // C2
  t3 = f(42); // C3
  X[i] = t1 + t2 + t3;
 }
}

int f(int v)
  return (v+1);
}
```



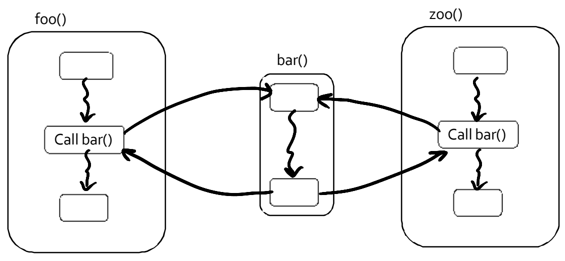
8

Solution Attempt #2

- Build a "supergraph" = inter-procedural CFG
- Replace each call from P to Q with
  - An edge from point before the call (call point) to Q's entry point
  - An edge from Q's exit point to the point after the call (return pt)
  - Add assignments of actuals to formals, and assignment of return value
- Good: efficient
  - Graph of each function included exactly once in the supergraph
  - Works for recursive functions (although local variables need additional treatment)
- Bad: imprecise, "context-insensitive"
  - The "unrealizable paths problem": dataflow facts can propagate along infeasible control paths

9

Unrealizable Paths



10

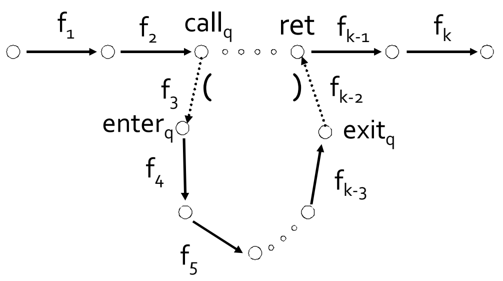
Interprocedural Analysis

```
begin
proc p() is¹
  [x := a + 1]²
end³
[a=7]⁴
[call p()]⁵₆
[print x]⁷
[a=9]⁸
[call p()]⁹₁₀
[print a]¹¹
end
```

- Extend language with begin/end and with $[\text{call } p()]^{clab}_{rlab}$
- Call label clab, and return label rlab

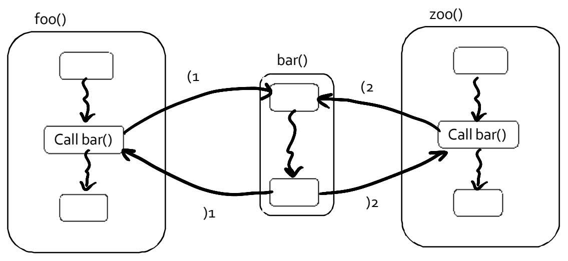
11

IVP: Interprocedural Valid Paths



- IVP: all paths with matching calls and returns
  - And prefixes

Valid Paths



Interprocedural Valid Paths

- **IVP** set of paths
  - Start at program entry
- Only considers matching calls and returns
  - aka, valid
- Can be defined via context free grammar
  - matched ::= matched $(_i$ matched $)_i$ | ε
  - valid ::= valid $(_i$ matched | matched
    - *paths* can be defined by a regular expression

The Join-Over-Valid-Paths (JVP)

- vpaths(n) all valid paths from program start to n
- JVP[n] $= \sqcup\{[\![e_1, e_2, \ldots, e]\!]$ (initial)
  $(e_1, e_2, \ldots, e) \in$ vpaths(n)}
- JVP $\sqsubseteq$ JFP
  - In some cases the JVP can be computed
  - (Distributive problem)

Sharir and Pnueli '82

- Call String approach
  - Blend interprocedural flow with intra procedural flow
  - Tag every dataflow fact with call history
- Functional approach
  - Determine the effect of a procedure
    - E.g., in/out map
  - Treat procedure invocations as "super ops"

The Call String Approach

- Record at every node a pair (l, c) where l ∈ L is the dataflow information and c is a suffix of unmatched calls
- Use Chaotic iterations
- To guarantee termination limit the size of c (typically 1 or 2)
- Emulates inline (but no code growth)
- Exponential in size of c

begin

proc p() is^{1}

[x := a + 1]^{2}

end^{3}

[a=7]^{4}

[call p()]$^{5}_{6}$

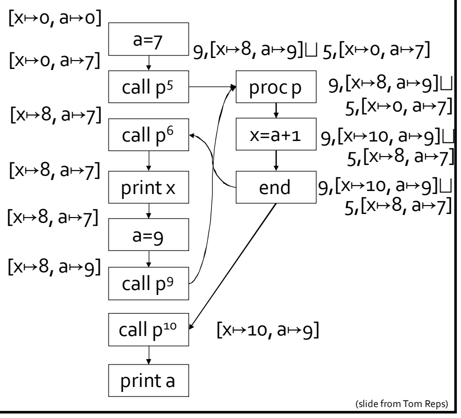
[print x]^{7}

[a=9]^{8}

[call p()]$^{9}_{10}$

[print a]^{11}

end



(slide from Tom Reps)

begin^{0}

proc p() is^{1}

if [b]^{2} then (

[a := a -1]^{3}

[call p()]$^{4}_{5}$

[a := a + 1]^{6}

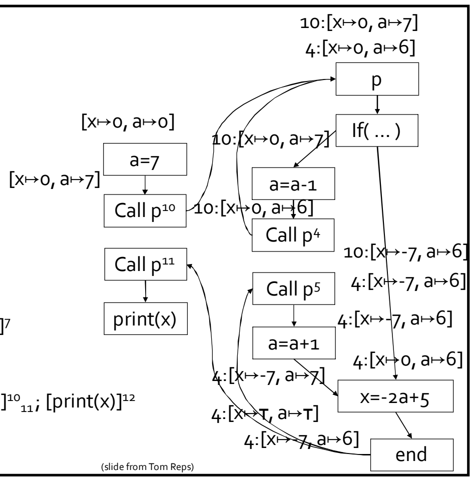
)

[x := -2* a + 5]^{7}

end^{8}

[a=7]^{9} ; [call p()]$^{10}_{11}$; [print(x)]^{12}

end^{13}



(slide from Tom Reps)

The Functional Approach

- The meaning of a procedure is mapping from states into states
- The abstract meaning of a procedure is function from an abstract state to abstract states

begin

proc p() is[1]

if $[b]^2$ then (

$[a := a - 1]^3$

$[\text{call } p()]^4_5$

$[a := a + 1]^6$

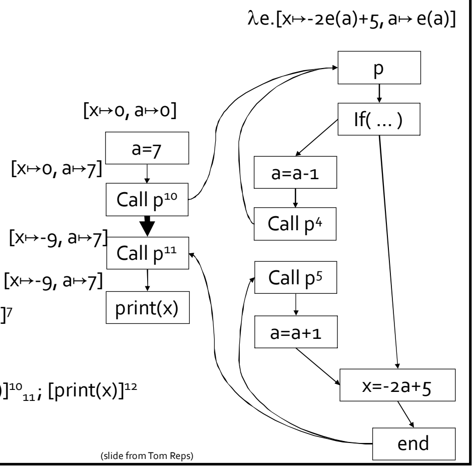
)

$[x := -2 * a + 5]^7$

end^8

$[a=7]^9$; $[\text{call } p()]^{10}_{11}$; $[\text{print}(x)]^{12}$

end



(slide from Tom Reps)

begin

proc p() is[1]

if $[b]^2$ then (

$[a := a - 1]^3$

$[\text{call } p()]^4_5$

$[a := a + 1]^6$

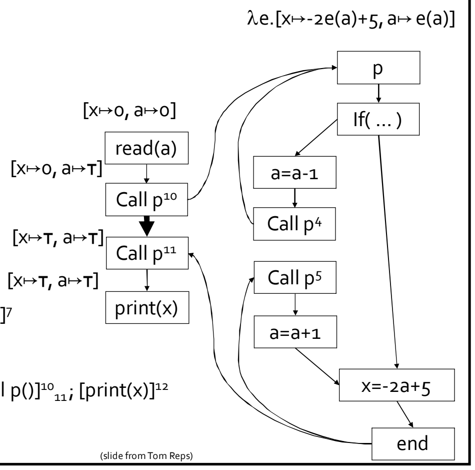
)

$[x := -2 * a + 5]^7$

end^8

$[\text{read}(a)]^9$; $[\text{call } p()]^{10}_{11}$; $[\text{print}(x)]^{12}$

end



(slide from Tom Reps)

Functional Approach: Main Idea

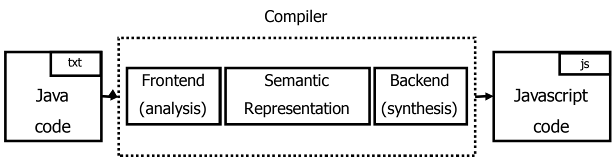
- Iterate on the abstract domain of functions from L to L
- Two phase algorithm
  - Compute the dataflow solution at the exit of a procedure as a function of the initial values at the procedure entry (functional values)
  - Compute the dataflow values at every point using the functional values
- Computes JVP for distributive problems

Meanwhile, IRL

- new compilers for new languages
- new compilers for old languages
  - e.g., Java->JavaScript
- new uses of compiler technology
- …

24

Google Web Toolkit



GWT Compiler Optimization

```
public class ShapeExample implements EntryPoint {
  private static final double SIDE_LEN_SMALL = 2;
  private final Shape shape = new SmallSquare();
  public static abstract class Shape {
    public abstract double getArea();
  }
  public static abstract class Square extends Shape {
    public double getArea() { return getSideLength() * getSideLength(); }
    public abstract double getSideLength();
  }
  public static class SmallSquare extends Square {
    public double getSideLength() { return SIDE_LEN_SMALL; }
  }
  public void onModuleLoad() {
    Shape shape = getShape();
    Window.alert("Area is " + shape.getArea());
  }
  private Shape getShape() { return shape; }
```

(source: http://dl.google.com/io/2009/pres/Th_1045_TheStoryofyourCompile-ReadingtheTeaLeavesoftheGWTCompilerforanOptimizedFuture.pdf)

GWT Compiler Optimization

```
public class ShapeExample implements EntryPoint {
  public void onModuleLoad() {
    Window.alert("Area is 4.0");
  }
```

(source: http://dl.google.com/io/2009/pres/Th_1045_TheStoryofyourCompile-ReadingtheTeaLeavesoftheGWTCompilerforanOptimizedFuture.pdf)

Adobe ActionScript

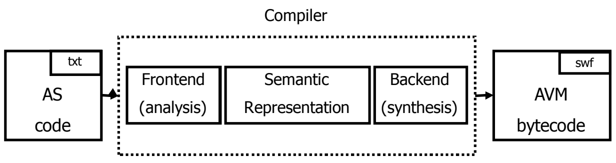
/*AS*/

First introduced in Flash Player 9, **ActionScript 3** is an object-oriented programming (OOP) language based on ECMAScript—the same standard that is the basis for JavaScript—and provides incredible gains in runtime performance and developer productivity. **ActionScript 2**, the version of ActionScript used in Flash Player 8 and earlier, continues to be supported in Flash Player 9 and Flash Player 10.

ActionScript 3.0 introduces a new highly optimized ActionScript Virtual Machine, AVM2, which dramatically exceeds the performance possible with AVM1. As a result, ActionScript 3.0 code executes up to 10 times faster than legacy ActionScript code.

(source: http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html)

Adobe ActionScript



Adobe ActionScript: Tamarin

The goal of the "Tamarin" project is to implement a high-performance, open source implementation of the ActionScript™ 3 language, which is based upon and extends ECMAScript 3rd edition (ES3). ActionScript provides many extensions to the ECMAScript language, including packages, namespaces, classes, and optional strict typing of variables.
"Tamarin" implements both a high-performance just-in-time compiler and interpreter.

The Tamarin virtual machine is used within the Adobe® Flash® Player and is also being adopted for use by projects outside Adobe. The Tamarin just-in-time compiler (the "NanoJIT") is a collaboratively developed component used by both Tamarin and Mozilla TraceMonkey. The ActionScript compiler is available as a component from the open source Flex SDK.

Mozilla SpiderMonkey

- SpiderMonkey is a fast interpreter
- runs an untyped bytecode and operates on values of type jsval—type-tagged double-sized values that represent the full range of JavaScript values.
- SpiderMonkey contains two JavaScript Just-In-Time (JIT) compilers, a garbage collector, code implementing the basic behavior of JavaScript values…

- SpiderMonkey's interpreter is mainly a single, tremendously long function that steps through the bytecode one instruction at a time, using a switch statement (or faster alternative, depending on the compiler) to jump to the appropriate chunk of code for the current instruction.

(source: https://developer.mozilla.org/En/SpiderMonkey/Internals)

Mozilla SpiderMonkey: Compiler

- consumes JavaScript source code
- produces a *script* which contains bytecode, source annotations, and a pool of string, number, and identifier literals. The script also contains objects, including any functions defined in the source code, each of which has its own, nested script.

- The compiler consists of
  - a random-logic rather than table-driven lexical scanner
  - a recursive-descent parser that produces an AST
  - a tree-walking code generator

- The emitter does some constant folding and a few codegen optimizations

(source: https://developer.mozilla.org/En/SpiderMonkey/Internals)

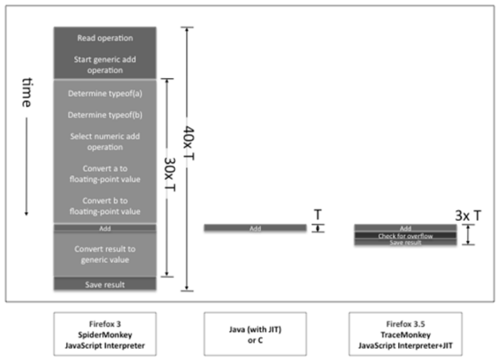
Mozilla SpiderMonkey

- SpiderMonkey contains a just-in-time trace compiler that converts bytecode to machine code for faster execution.
- The JIT works by detecting commonly executed loops, tracing the executed bytecodes in those loops as they run in the interpreter, and then compiling the trace to machine code.
- See the page about the Tracing JIT for more details.

- The SpiderMonkey GC is a mark-and-sweep, non-conservative (exact) collector.

(source: https://developer.mozilla.org/En/SpiderMonkey/Internals) 33

Mozilla TraceMonkey


time
Read operation
Start generic add operation
Determine typeof(a)
Determine typeof(b)
Select numeric add operation
Convert a to floating-point value
Convert b to floating-point value
Add
Convert result to generic value
Save result
40x T
30x T
Add
T
Add
Check for overflow
Save result
3x T
Firefox 3 SpiderMonkey JavaScript Interpreter
Java (with JIT) or C
Firefox 3.5 TraceMonkey JavaScript Interpreter+JIT

(source: http://hacks.mozilla.org/2009/07/tracemonkey-overview/) 34

Mozilla TraceMonkey

- Goal: generate type-specialized code
- Challenges
  - no type declarations
  - statically trying to determine types mostly hopeless

- Idea
  - run the program for a while and observe types
  - use observed types to generate type-specialized code
  - compile traces

- Sounds suspicious?

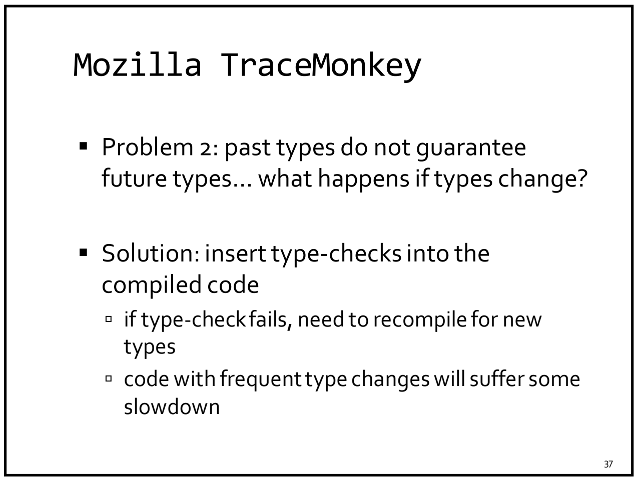
35

Mozilla TraceMonkey

- Problem 1: "observing types" + compiling possibly more expensive than running the code in the interpreter

- Solution: only compile code that executes many times
  - "hot code" = loops
  - initially everything runs in the interpreter
  - start tracing a loop once it becomes "hot"

36

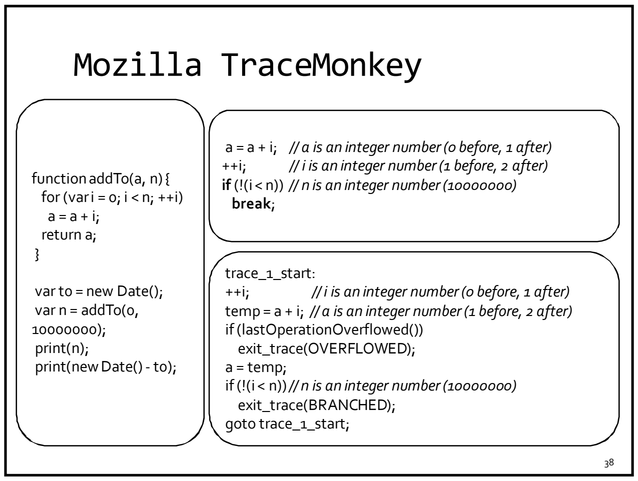


Mozilla TraceMonkey

- Problem 2: past types do not guarantee future types… what happens if types change?

- Solution: insert type-checks into the compiled code
  - if type-check fails, need to recompile for new types
  - code with frequent type changes will suffer some slowdown

37



Mozilla TraceMonkey

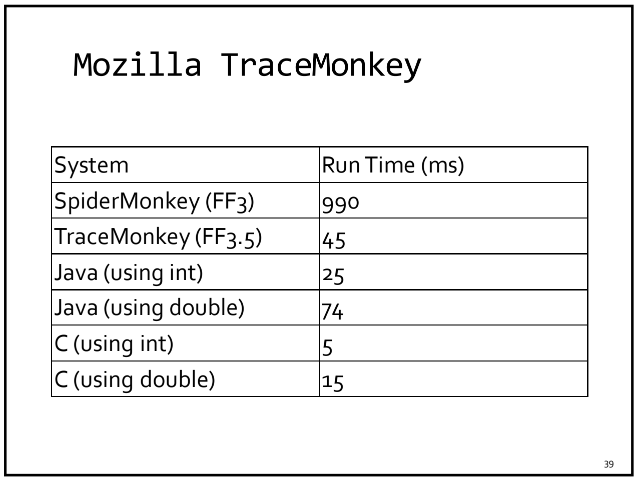
```
function addTo(a, n) {
  for (var i = 0; i < n; ++i)
    a = a + i;
  return a;
}

var to = new Date();
var n = addTo(0,
10000000);
print(n);
print(new Date() - to);
```

```
a = a + i;   // a is an integer number (0 before, 1 after)
++i;         // i is an integer number (1 before, 2 after)
if (!(i < n)) // n is an integer number (10000000)
  break;
```

```
trace_1_start:
++i;            // i is an integer number (0 before, 1 after)
temp = a + i;  // a is an integer number (1 before, 2 after)
if (lastOperationOverflowed())
   exit_trace(OVERFLOWED);
a = temp;
if (!(i < n)) // n is an integer number (10000000)
   exit_trace(BRANCHED);
goto trace_1_start;
```

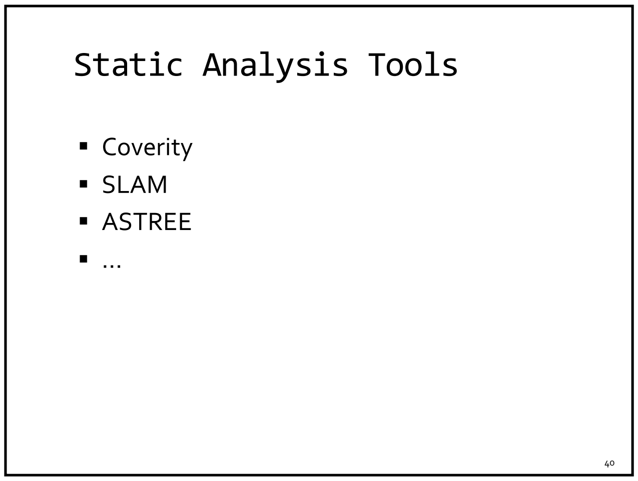
38



Mozilla TraceMonkey

System	Run Time (ms)
SpiderMonkey (FF3)	990
TraceMonkey (FF3.5)	45
Java (using int)	25
Java (using double)	74
C (using int)	5
C (using double)	15

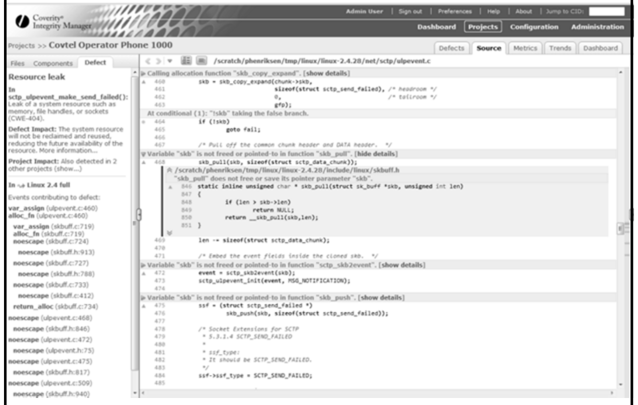
39



Static Analysis Tools

- Coverity
- SLAM
- ASTREE
- …

40



Projects >> Covtel Operator Phone 1000

Resource leak

In sctp_ulpevent_make_send_failed(): Leak of a system resource such as memory, file handles, or sockets (CWE-404).

Defect Impact: The system resource will not be reclaimed and reused, reducing the future availability of the resource. More information...

Project Impact: Also detected in 2 other projects (show...)

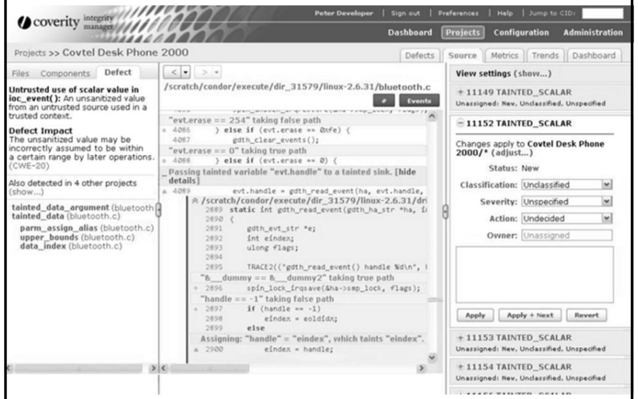
In Linux 2.4 full

Events contributing to defect:

var_assign (ulpevent.c:460)
alloc_fn (ulpevent.c:460)
  var_assign (skbuff.c:719)
  alloc_fn (skbuff.c:719)
  noescape (skbuff.c:724)
    noescape (skbuff.h:913)
  noescape (skbuff.c:727)
    noescape (skbuff.h:788)
  noescape (skbuff.c:733)
    noescape (skbuff.c:412)
  return_alloc (skbuff.c:734)
noescape (ulpevent.c:468)
  noescape (skbuff.h:846)
noescape (ulpevent.c:472)
  noescape (ulpevent.h:75)
noescape (ulpevent.c:475)
  noescape (skbuff.h:817)
noescape (ulpevent.c:509)
  noescape (skbuff.h:940)

/scratch/phenriksen/tmp/linux/linux-2.4.28/net/sctp/ulpevent.c

```
Calling allocation function "skb_copy_expand". [show details]
460         skb = skb_copy_expand(chunk->skb,
461                               sizeof(struct sctp_send_failed), /* headroom */
462                               0,                               /* tailroom */
463                               gfp);
At conditional (1): "!skb" taking the false branch.
464         if (!skb)
465                 goto fail;
466
467         /* Pull off the common chunk header and DATA header.  */
Variable "skb" is not freed or pointed-to in function "skb_pull". [hide details]
468         skb_pull(skb, sizeof(struct sctp_data_chunk));
    /scratch/phenriksen/tmp/linux/linux-2.4.28/include/linux/skbuff.h
    "skb_pull" does not free or save its pointer parameter "skb".
    846 static inline unsigned char * skb_pull(struct sk_buff *skb, unsigned int len)
    847 {
    848         if (len > skb->len)
    849                 return NULL;
    850         return __skb_pull(skb,len);
    851 }
469         len -= sizeof(struct sctp_data_chunk);
470
471         /* Embed the event fields inside the cloned skb.  */
Variable "skb" is not freed or pointed-to in function "sctp_skb2event". [show details]
472         event = sctp_skb2event(skb);
473         sctp_ulpevent_init(event, MSG_NOTIFICATION);
474
Variable "skb" is not freed or pointed-to in function "skb_push". [show details]
475         ssf = (struct sctp_send_failed *)
476                 skb_push(skb, sizeof(struct sctp_send_failed));
477
478         /* Socket Extensions for SCTP
479          * 5.3.1.4 SCTP_SEND_FAILED
480          *
481          * ssf_type:
482          * It should be SCTP_SEND_FAILED.
483          */
484         ssf->ssf_type = SCTP_SEND_FAILED;
485
```



Projects >> Covtel Desk Phone 2000

Untrusted use of scalar value in ioc_event(): An unsanitized value from an untrusted source used in a trusted context.

Defect Impact
The unsanitized value may be incorrectly assumed to be within a certain range by later operations. (CWE-20)

Also detected in 4 other projects (show...)

tainted_data_argument (bluetooth
tainted_data (bluetooth.c)
  parm_assign_alias (bluetooth.c)
  upper_bounds (bluetooth.c)
  data_index (bluetooth.c)

/scratch/condor/execute/dir_31579/linux-2.6.31/bluetooth.c

```
"evt.erase == 254" taking false path
        } else if (evt.erase == 0xfe) {
            gdth_clear_events();
"evt.erase == 0" taking true path
        } else if (evt.erase == 0) {
_ Passing tainted variable "evt.handle" to a tainted sink. [hide
details]
            evt.handle = gdth_read_event(ha, evt.handle,
    /scratch/condor/execute/dir_31579/linux-2.6.31/dri
    static int gdth_read_event(gdth_ha_str *ha, i
    {
        gdth_evt_str *e;
        int eindex;
        ulong flags;

        TRACE2(("gdth_read_event() handle %d\n",
    "&__dummy == &__dummy2" taking true path
        spin_lock_irqsave(&ha->smp_lock, flags);
    "handle == -1" taking false path
        if (handle == -1)
            eindex = eoldidx;
        else
    Assigning: "handle" = "eindex", which taints "eindex".
            eindex = handle;
```

View settings (show...)

+ **11149 TAINTED_SCALAR**
Unassigned: New, Unclassified, Unspecified

− **11152 TAINTED_SCALAR**

Changes apply to **Covtel Desk Phone 2000/*** (adjust...)

Status: New
Classification: Unclassified
Severity: Unspecified
Action: Undecided
Owner: Unassigned

Apply | Apply + Next | Revert

+ **11153 TAINTED_SCALAR**
Unassigned: New, Unclassified, Unspecified

+ **11154 TAINTED_SCALAR**
Unassigned: New, Unclassified, Unspecified

Lots and lots of research

- Program Analysis
- Program Synthesis
- …
- Next semester
  - Project – contact me if you're interested
  - Advanced course in program analysis

THE END