

SERVER-SIDE ANALYSIS

Ben Livshits, Microsoft Research

Overview of Today's Lecture

2

- Static analysis for bug finding
- Scripting languages analyzed (UsenixSec '05 paper)
- Runtime analysis
 - ▣ Fuzzing
 - ▣ Pen testing
 - ▣ Tainting
 - ▣ Symbolic execution

Compilers Under the Hood

3

```
File Edit Options Buffers Tools C Help
int main()
{
    printf("hello,world!\r\n");

    return (0);
}

--UU-:----F1 123.c      All L1      (C/l Abbrev)-----
--* mode: compilation; default-directory: "/home/dodolook/" --*
Compilation started at Mon May 30 15:36:27

gcc -o hello 123.c
123.c: In function main:
123.c:3:3: warning: incompatible implicit declaration of built-in function printf
Compilation finished at Mon May 30 15:36:29

-UUU:~*--F1 *compilation* All L1 (Compilation:exit [0])-----
Compilation finished
```

Stages of Compilation

4

Source code

Lexing

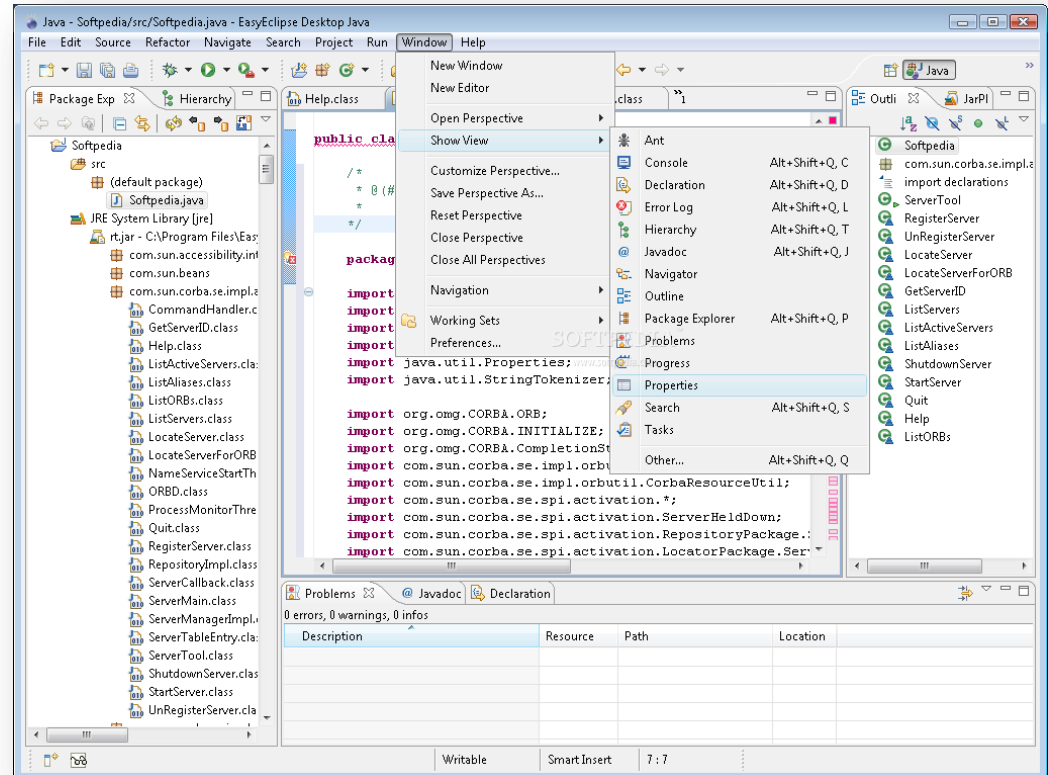
Parsing

IR

Analysis

Code generation

Executable code



Stages of Compilation

5

Source code

Lexing

Parsing

IR

Analysis

Code generation

Executable code

```
addlinenum + (~/.Documents/compiler/lex) - GVIM
File Edit Tools Syntax Buffers Window Help
[Icons]
/** Definition section **/
%{
/* C code to be copied verbatim */
#include <stdio.h>
%}

/* This tells flex to read only one input file */
%option noyywrap

*** Rules section ***

/* [0-9]+ matches a string of one or more digits */
[0-9]+ {
    /* yytext is a string containing the matched text. */
    printf("Saw an integer: %s\\n", yytext);
}

. { /* Ignore all other characters. */ }

*** C Code section ***

int main(void)
{
    /* Call the lexer, then quit. */
    yylex();
    return 0;
}
```

29,5 All

Stages of Compilation

6

Source code

Lexing

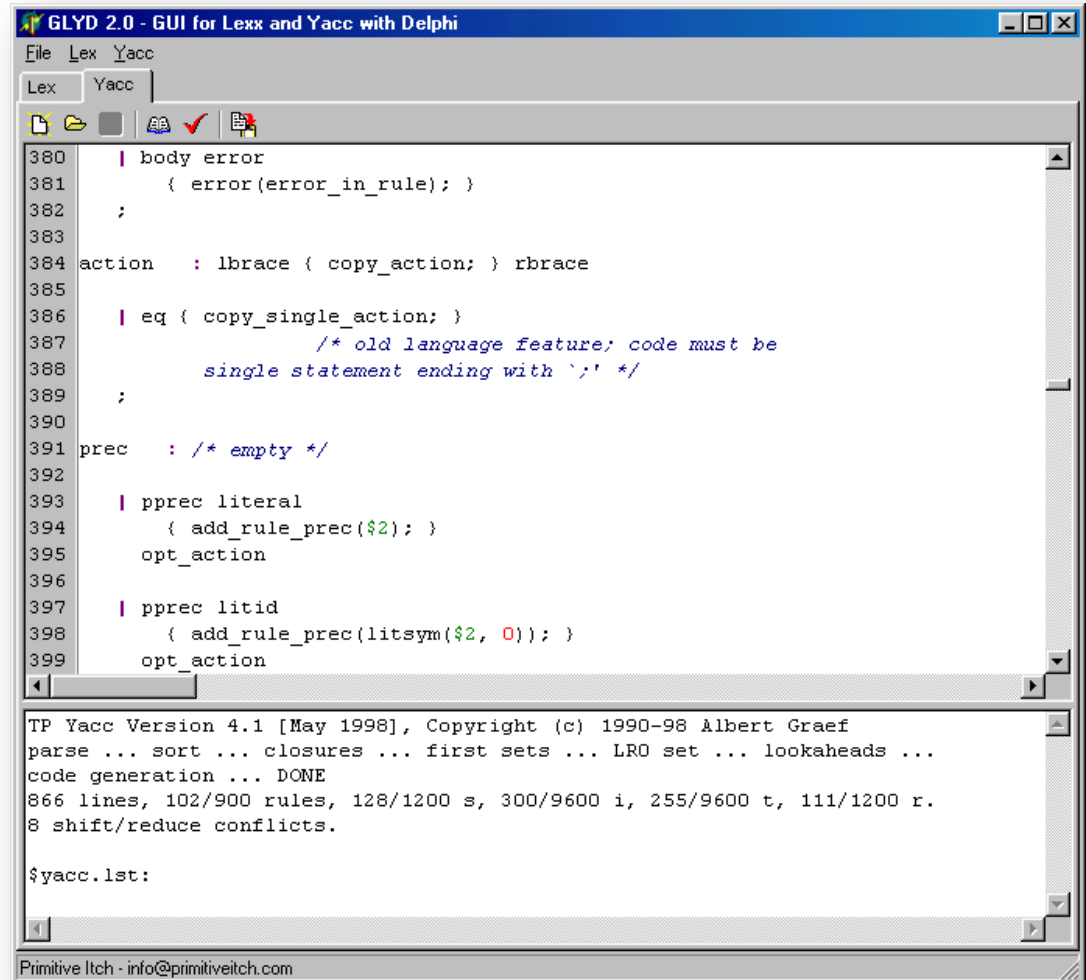
Parsing

IR

Analysis

Code generation

Executable code



```
380 | body error
381 |   { error(error_in_rule); }
382 |   ;
383
384 action   : lbrace { copy_action; } rbrace
385
386 | eq { copy_single_action; }
387 |   /* old language feature; code must be
388 |   single statement ending with ';' */
389 |   ;
390
391 prec    : /* empty */
392
393 | pprec literal
394 |   { add_rule_prec($2); }
395 |   opt_action
396
397 | pprec litid
398 |   { add_rule_prec(litsym($2, 0)); }
399 |   opt_action
```

TP Yacc Version 4.1 [May 1998], Copyright (c) 1990-98 Albert Graef
parse ... sort ... closures ... first sets ... LRO set ... lookaheads ...
code generation ... DONE
866 lines, 102/900 rules, 128/1200 s, 300/9600 i, 255/9600 t, 111/1200 r.
8 shift/reduce conflicts.

\$yacc.lst:

Primitive Itch - info@primitiveitch.com

Stages of Compilation

7

Source code

Lexing

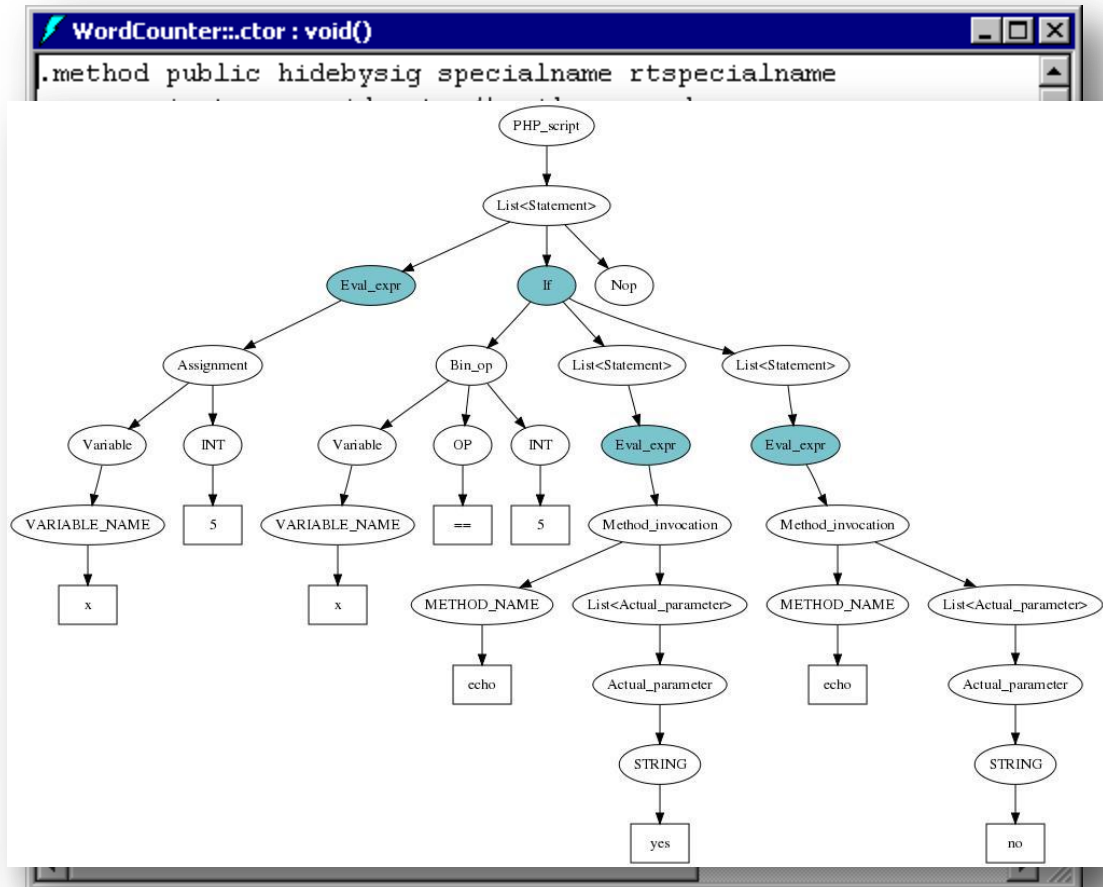
Parsing

IR

Analysis

Code generation

Executable code



Stages of Compilation

8

Source code

Lexing

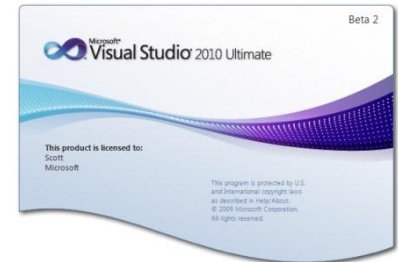
Parsing

IR

Analysis

Code generation

Executable code



Klocwork.



Stages of Compilation

9

Source code

Lexing

Parsing

IR

Analysis

Code generation

Executable code



The screenshot shows a web browser window with the URL `http://rise4fun.c...` and the page title `Vcc @ RiSE4fun - A Ve..`. The main heading is **RiSE4fun**. Below the heading, it says *gave 116,689 answers!* and *Click on a tool to load a sample then ask!*. There are several buttons for different tools: `agl`, `bek`, `boogie`, `code contracts`, `concurrent`, `revisions`, `dafny`, `dkal`, `esm`, `f*`, `formula`, `heapdbg`, `poirot`, `pex`, `rex`, `slayer`, `spec#`, `vcc`, and `z3`. The `vcc` button is highlighted in purple. Below the buttons is a text area containing the following C code:

```
#include <vcc.h>

int main()
{
    int x, y;
    _(assert x >= y)
    return 0;
}
```

Below the code area is a button labeled `ask vcc` and a question: *Does this C program always work? Click 'ask vcc'! Read more or watch the video.* At the bottom right, there are social media sharing buttons for `Tweet` and `Like` (with a count of 151).

Static Analysis

10

- Pros?

- Cons?

Static Analysis Tool for Bug Finding: Plan

11

1. Read the program
2. Transform into an Intermediate Representation (IR)
3. Do analysis on the IR
4. Output results

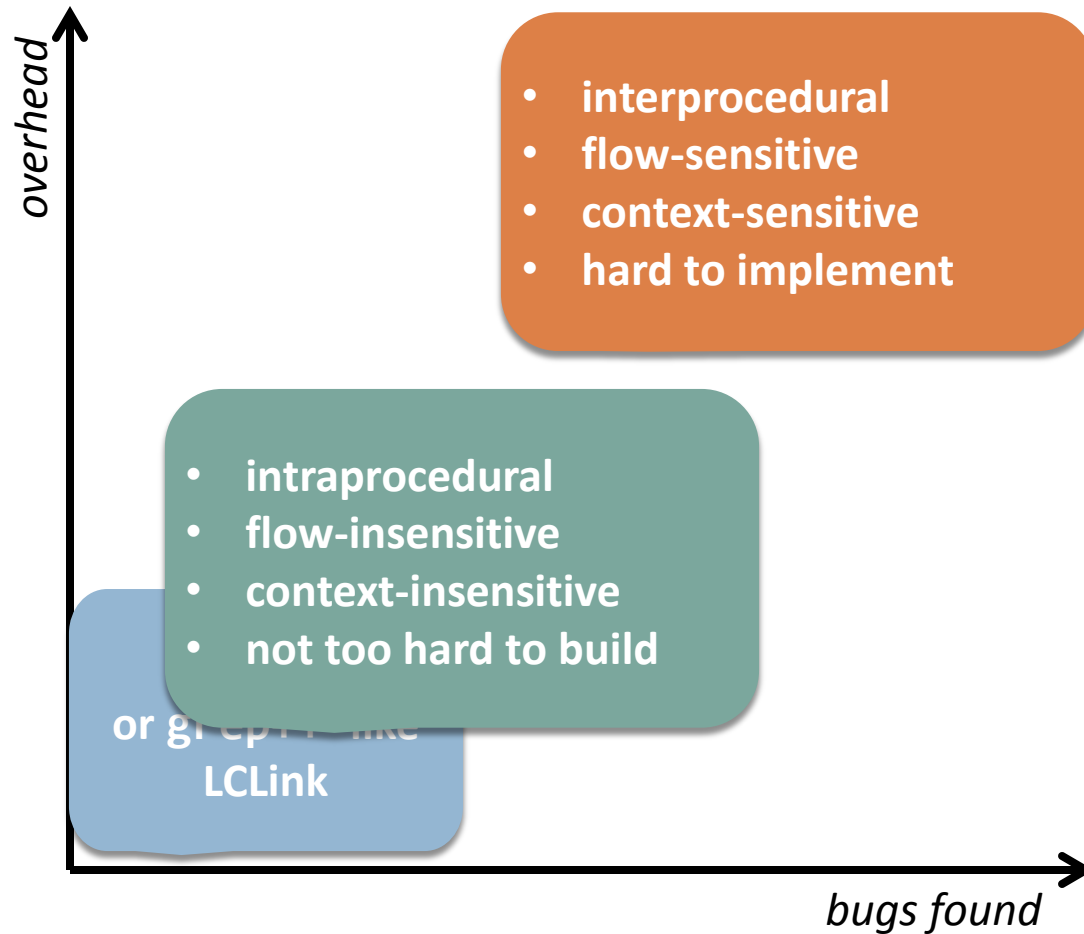
Dimensions of Analysis

12

- Intraprocedural vs. interprocedural
- Flow sensitive vs. flow-insensitive
- Context sensitive vs. context-insensitive

Cost vs. Effectiveness

13



A static analyzer for finding dynamic programming errors



William R. Bush*, Jonathan D. Pincus and David J. Sielaff

Intrinsa Corporation, Mountain View, CA, U.S.A.

SUMMARY

There are important classes of programming errors that are hard to diagnose, both manually and automatically, because they involve a program's dynamic behavior. This article describes a compile-time analyzer that detects these dynamic errors in large, real-world programs. The analyzer traces execution paths through the source code, modeling memory and reporting inconsistencies. In addition to avoiding false paths through the program, this approach provides valuable contextual information to the programmer who needs to understand and repair the defects. Automatically-created models, abstracting the behavior of individual functions, allow inter-procedural defects to be detected efficiently. A product built on these techniques has been used effectively on several large commercial programs. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: program analysis; program error checking

INTRODUCTION

There are important classes of programming errors that are hard to diagnose, both manually and automatically, because they involve a program's dynamic behavior. They include invalid pointer references, faulty storage allocation, the use of uninitialized memory, and improper operations on resources such as files (trying to close a file that is already closed, for example).

Finding and fixing such errors is difficult and expensive. They are usually found late in the development process. Extensive testing is often needed to find them, because they are commonly caused by complex interactions between components. Our measurements indicate that in commercial C and C++ code, on the order of 90% of these errors are caused by the interaction of multiple functions. In addition, problems may be revealed only in error conditions or other unusual situations, which are difficult to provoke by standard testing methods.

Traditional checking provided by the error-checking portion of compilers identifies errors relating to the static expression of a program, such as syntax errors, type violations, and mismatches between

*Correspondence to: William R. Bush, 1739 Lexington Avenue, San Mateo, CA 94402-4024, U.S.A.

Historical background

- Intrinsa
 - ▣ 1997-200?
 - ▣ paved way for MS
- Coverity
 - ▣ Out of Stanford
 - ▣ Commercial static analysis tools
- Fortify
 - ▣ Tools for security
- Klockwork

Paper Contributions

15

- Interprocedural static analysis algorithm
 - ▣ Address dynamic language features
 - ▣ Hash table use
 - ▣ Regular expression matching

- Features
 - ▣ Symbolic execution inside basic blocks
 - ▣ Basic block summaries

Paper Contributions

16

- Focus
 - ▣ SQL injection vulnerabilities. Why? Good idea?
 - ▣ XSS – claim to handle with minor modifications

- Experiments
 - ▣ 6 PHP apps
 - ▣ Finds 105 previously unknown vulnerabilities

PHP Language Features

17

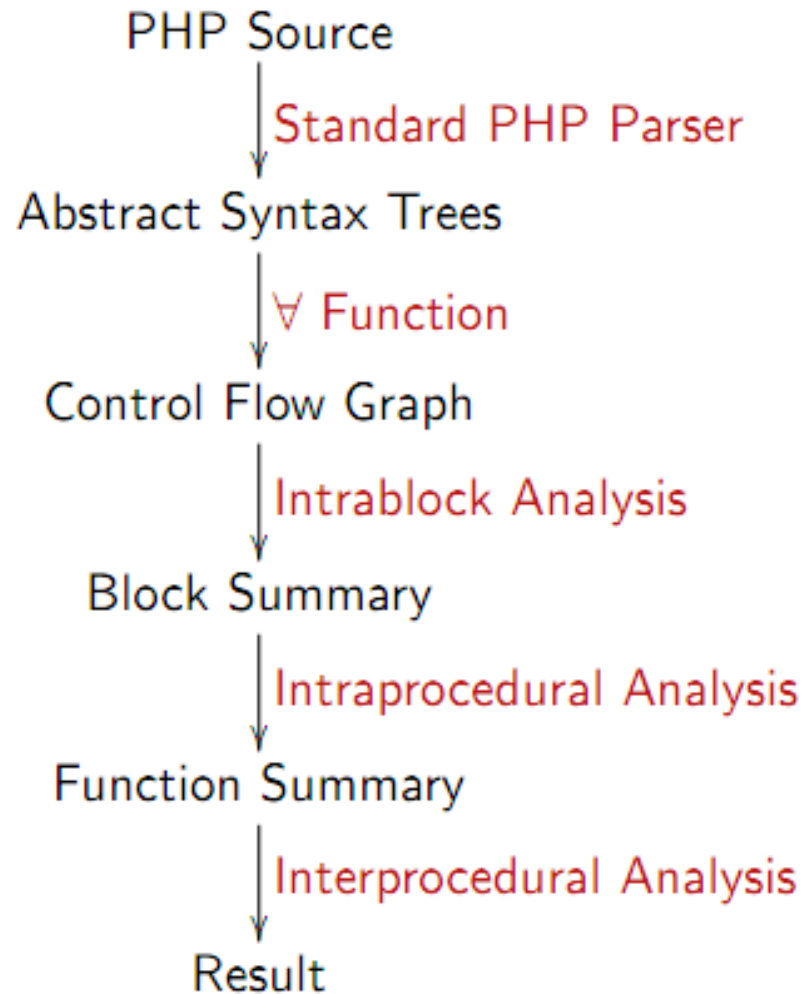
- Natural SQL integration
 - ▣ `$rows = mysql_query("UPDATE users SET pass='$pass' WHERE userid='$userid');`

- Dynamic types and implicit casts
 - ▣ `if ($userid < 0) exit;`
 - ▣ `$query = "SELECT * from users WHERE userid='$userid';"`

- Global environment
 - ▣ `$_GET['name']` or `$name`
 - ▣ `$` used with `register_globals = on`? Attacker may provide arbitrary value for `$superuser` by inserting something like `$superuser=1` into HTTP request

Analysis Steps (Section 3)

18



Basic blocks: Simulation

19

- Build up a model mapping labels -> values
- Special treatment of strings. Why?
- Special treatment of (some) booleans. Why?

Various Data Types: Representation

20

Strings Most fundamental type

- ▶ Concatenation of string segments
- ▶ `contains(σ)`: String with substrings from a set σ of memory locations

Basic Block Summary

Set	Symbol	Description
Error set	E	Input variables which must be sanitized before entering this basic block
Return value	R	Representation for return value
Untaint set	U	Sanitized locations for each successor
Termination predicate	T	Block contains <code>exit()</code> or calls another termination function
Value flow	F	Set of location pairs (l_1, l_2) where l_1 is a substring of l_2 on exit
Definitions	D	Defined memory locations

Function Summary

22

Set	Symbol	
Error set	E	Input
Return value	R	Repre
Sanitized values	S	Saniti
Program exit	X	Block termi

Memory location that can flow to database inputs

for main function, this cannot include
`$_GET[...]` or `$_POST[...]`

Function Summary

23

Set	Symbol	
Error set	E	Input ent
Return value	R	
Sanitized values	S	Saniti
Program exit	X	Block termi

string-typed parameters or globals that might be returned, either fully or as part of a longer string

```
function make query($user, $pass) {  
  global $table;  
  return "SELECT * from $table "  
    "where user = $user and  
      pass = $pass";  
}
```

```
 $R = \{ \$table, \$arg\#1, \$arg\#2 \}$ 
```

Function Summary

24

Set	Symbol	
Error set	E	Input ent
Return value	R	
Sanitized values	S	Saniti
Program exit	X	Block termi

the set of parameters or global variables that are sanitized on function exit

```
function is_valid($x) {  
    if (is_numeric($x)) return true;  
    return false;  
}
```

```
 $S = (\text{false} \Rightarrow \{\}, \text{true} \Rightarrow \{\text{arg\#1}\})$ 
```


Function Summary

25

Set	Symbol	Description
Error set	E	Input variables which must be sanitized before entering this basic block
Return value	R	Representation for return value
Sanitized values	S	Sanitized values
Program exit	X	Block termination

a Boolean which indicates whether the current function terminates program execution on all paths

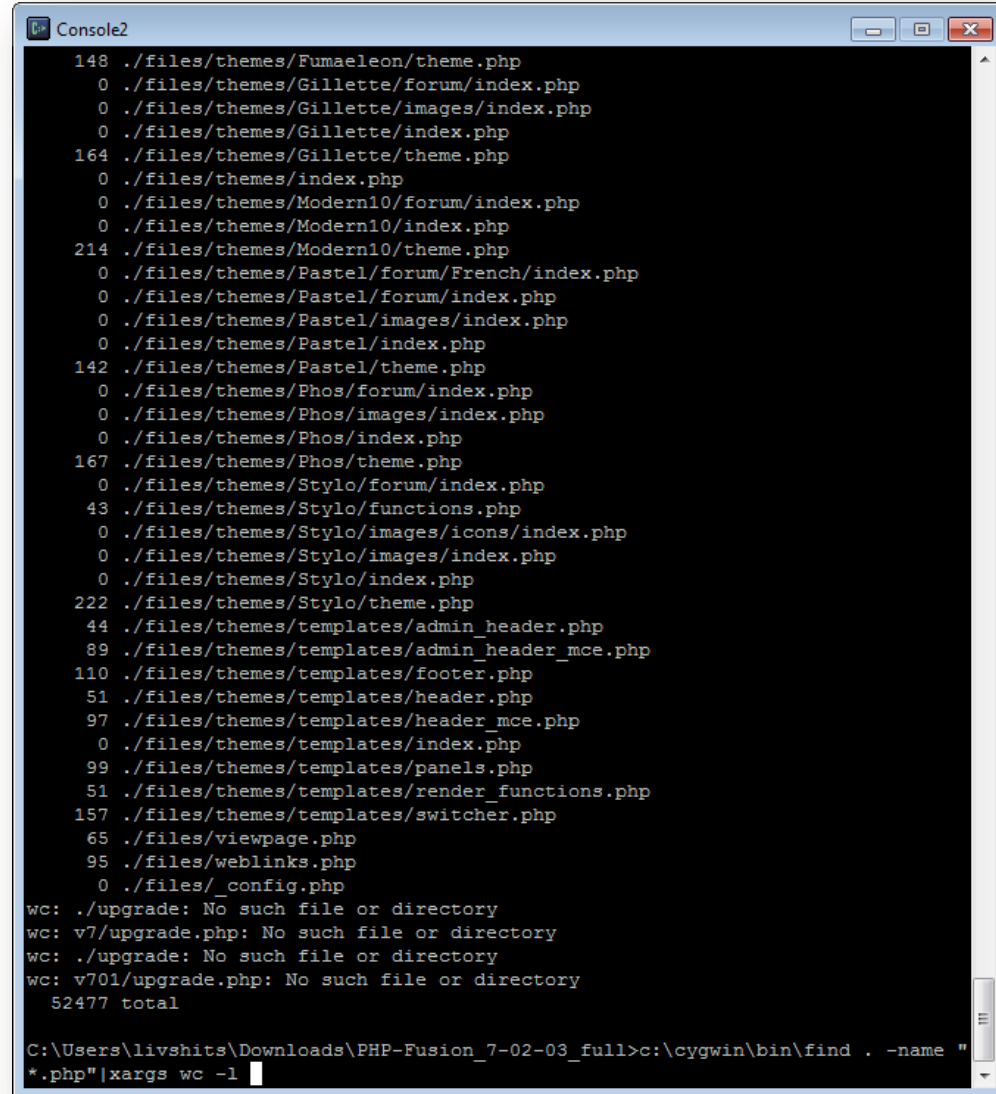
Interprocedural Analysis

Since we require the summary information of a function before we can analyze its callers, the order in which functions are analyzed is important. Due to the dynamic nature of PHP (e.g., include statements), we analyze functions on demand—a function f is analyzed and summarized when we first encounter a call to f . The summary is then memoized to avoid redundant analysis. Recursive function calls are rare in PHP programs. If we encounter a cycle during the analysis, our current implementation uses a dummy “no-op” summary as a model for the second invocation.

Why On Demand?

27

- ❑ PHP Fusion
- ❑ version 7-02-03
- ❑ about 52K lines of code
- ❑ But really only about 16,000 matter



```
Console2
148 ./files/themes/Fumaeleon/theme.php
0 ./files/themes/Gillette/forum/index.php
0 ./files/themes/Gillette/images/index.php
0 ./files/themes/Gillette/index.php
164 ./files/themes/Gillette/theme.php
0 ./files/themes/index.php
0 ./files/themes/Modern10/forum/index.php
0 ./files/themes/Modern10/index.php
214 ./files/themes/Modern10/theme.php
0 ./files/themes/Pastel/forum/French/index.php
0 ./files/themes/Pastel/forum/index.php
0 ./files/themes/Pastel/images/index.php
0 ./files/themes/Pastel/index.php
142 ./files/themes/Pastel/theme.php
0 ./files/themes/Phos/forum/index.php
0 ./files/themes/Phos/images/index.php
0 ./files/themes/Phos/index.php
167 ./files/themes/Phos/theme.php
0 ./files/themes/Stylo/forum/index.php
43 ./files/themes/Stylo/functions.php
0 ./files/themes/Stylo/images/icons/index.php
0 ./files/themes/Stylo/images/index.php
0 ./files/themes/Stylo/index.php
222 ./files/themes/Stylo/theme.php
44 ./files/themes/templates/admin_header.php
89 ./files/themes/templates/admin_header_mce.php
110 ./files/themes/templates/footer.php
51 ./files/themes/templates/header.php
97 ./files/themes/templates/header_mce.php
0 ./files/themes/templates/index.php
99 ./files/themes/templates/panels.php
51 ./files/themes/templates/render_functions.php
157 ./files/themes/templates/switcher.php
65 ./files/viewpage.php
95 ./files/weblinks.php
0 ./files/_config.php
wc: ./upgrade: No such file or directory
wc: v7/upgrade.php: No such file or directory
wc: ./upgrade: No such file or directory
wc: v701/upgrade.php: No such file or directory
52477 total

C:\Users\livshits\Downloads\PHP-Fusion_7-02-03_full>c:\cygwin\bin\find . -name "
*.php"|xargs wc -l
```

Checker Input

28

- We seed the checker with a small set of query functions (e.g. `mysql_query`) and sanitization operations (e.g. `is_numeric`).
- The checker infers the rest automatically

Checker Output

29

- Errors
 - ▣ Variables controlled by the attacker `$_GET[...]` and `$_POST[...]`

- Warnings
 - ▣ Other environment-define variables at the level of main

Result Summary

	Err Msgs	Bugs (FP)	Warn
e107	16	16 (0)	23
News Pro	8	8 (0)	8
myBloggie	16	16 (0)	23
DCP Portal	39	39 (0)	55
PHP Webthings	20	20 (0)	6
Total	99	99 (0)	115

Table 1: Summary of experiments. Err Msgs: number of reported errors. Bugs: number of confirmed bugs from error reports. FP: number of false positives. Warn: number of unique warning messages for variables of unresolved origin (uninspected).

31

question of the day


Are the techniques in the paper sound, i.e. do they find all SQL injection bugs?

Runtime Analysis Overview

32

- Black
- Fuz
- Per

analysis
execution

 <p>David Litchfield, Chris Anley, John Heasman, Bill Grindlay</p>	 <p>David Litchfield, Chris Anley</p>	 <p>Chris Anley, John Heasman</p>
 <p>Dafydd Stuttard, Marcus Pinto</p>	 <p>David Litchfield</p>	 <p>David Litchfield, Bill Grindlay</p>
 <p>Barrie Dempster</p>	 <p>Barrie Dempster</p>	 <p>Barrie Dempster</p>

Fuzzing: A Definition

33

“**Fuzz testing** or **fuzzing** is a software testing technique that provides invalid, unexpected, or random data to the inputs of a program. If the program fails (for example, by crashing or failing built-in code assertions), the defects can be noted.”

Wikipedia

Why Fuzz in General?

- Another point of view of testing
- If its automated, why not?
- Some Fuzzing Successes:
 - ▣ Apple Wireless flaw DoS (MOKB-30-11-2006)
 - ▣ Month of Browser Bugs in 2006, many found with input fuzzing:
 - IE: 25
 - Safari: 2
 - Firefox: 2
 - Opera: 1
 - Konquerer: 1

Need a Fuzzing Specification

35

```
setup "Webapp" do
  @host = "10.0.0.2"
  @port = 3000
  @headers = "HTTP_ACCEPT_CHARSET" => "utf-8 +"
```

What do they look for?

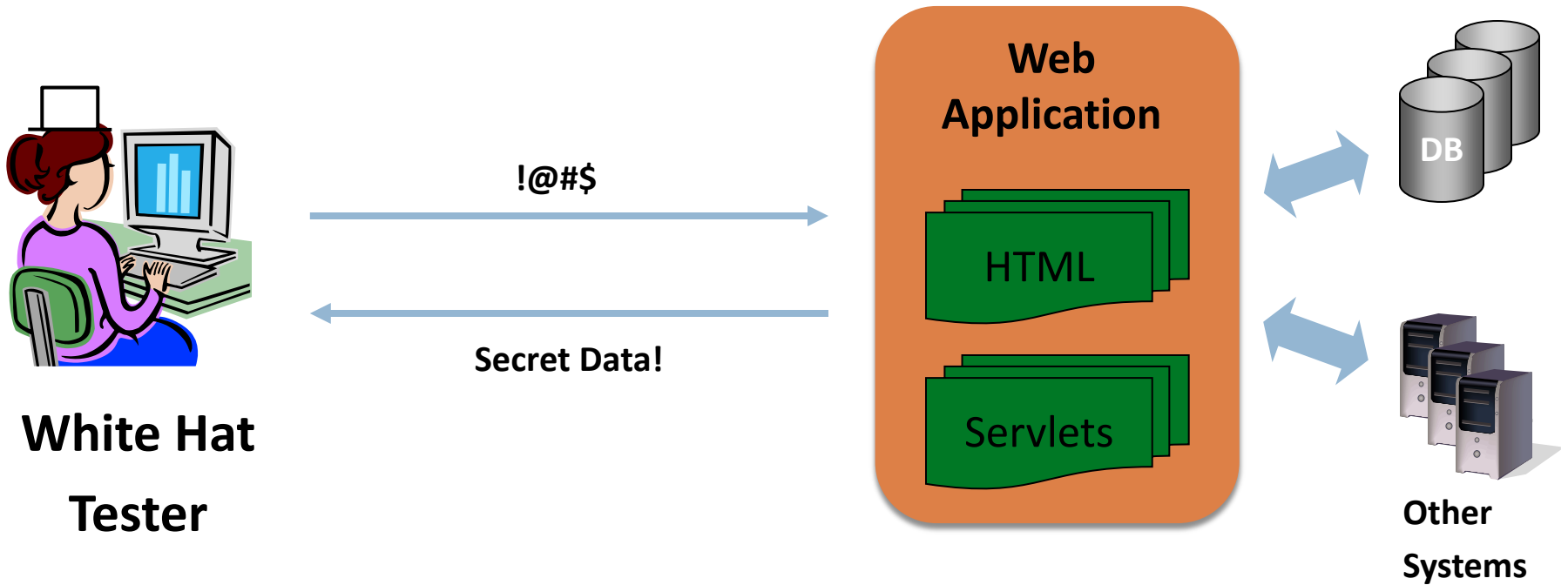
```
  attack "search-box" do
    many :get, "/search.php", :query => { :q => word }
    many :get, "/search.php", :query => { :q => word }
  end

  attack "post-page" do
    once :get, "/login.php", :query =>
      { :user => :admin, :pass => :admin }
    many :post, "/post.php", :query =>
      { :title => word, :body => byte(50) }
  end
end
```

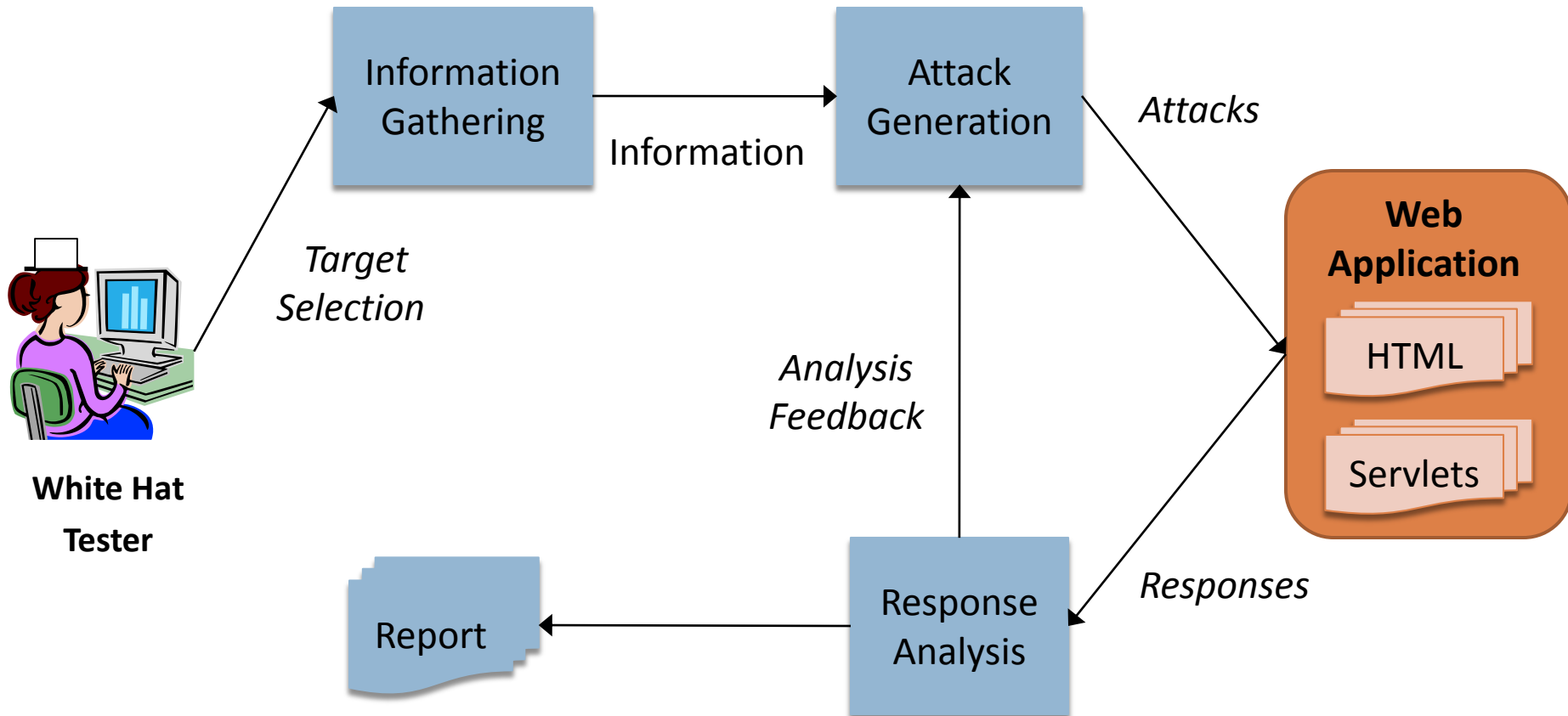
Fuzz testing of web applications, Hammersland and Snekkenes

Penetration Testing Overview

36



Penetration Testing: Phases



Tainting

38

- Negative tainting
 - Mark *or taint* untrusted input data at runtime
 - Stop execution when untrusted input reaches “sinks”
- Positive tainting
 - Taint trusted data such as constant strings only
 - Stop execution when data reaching “sinks” is not tainted
- Propagate the taint through at the application executes

```
String s =  
    req.getParameter("userName");  
String s2 = "hello" + s;  
output.println("<div>");  
output.println(s2);  
output.println("</div>");
```

Questions About Tainting

39

- How do we identify all sources in negative tainting?
- How do we remote taint?
- What is the runtime overhead?

Symbolic Execution

40

- Treat input values *symbolically*
- Propagate symbolic values through
- When encountering a conditional, consider both branches
- Use a *theorem prover* to eliminate infeasible paths

```
String s;  
if (!P) {  
    s = req.getParameter("userName");  
} else {  
    s = "";  
}
```

```
String s2 = "hello" + s;  
if (P) {  
    output.println("<div>");  
    output.println(s2);  
    output.println("</div>");  
} else {  
    output.println("hello");  
}
```


Summary

41

- Static analysis for bug finding
- Scripting languages analyzed (UsenixSec '05 paper)
- Runtime analysis
 - Black-box
 - ▣ Fuzzing
 - ▣ Pen testing
 - White-box
 - ▣ Tainting
 - ▣ Symbolic execution