

COMP 116: Analysis

⑥ Assignment 5

① Assignment 4 postmortum

analysis

static
no execution of program, full coverage, rule-based

dynamic
system execution

- trial and error
- detect dependencies
- deal w/ real runtime variables

- based on automated tests user interaction
- No guarantee of full coverage of source
- Ex: Valgrind

code

white box

lint, coverage, fortify, grep

binary

Veracode

black box

Techniques

① Data flow analysis

- Collect runtime info about data while in a static state

- ↳ Basic block (the code)
- ↳ Control flow
- ↳ Control path

② Control graph

- ↳ Node \Rightarrow block
- ↳ Edges \Rightarrow jumps / paths

③ Taint analysis (also DFA)
Identify variables that have
been tainted; uses vuln fns
Known as sink

④ Lexical analysis
code \Rightarrow tokens `gets(buf);`
 `/* ... gets */`

Strengths, weaknesses

- ⊕ find vulns w/ high confidence
- ⊖ false positives, false negatives
- ⊖ can't find config issues
- ⊖ can you prove they are vulns?

More on tainting (Perl, Ruby)

- Anything derived from tainted
data is tainted

```
i = getinput();  
two = 2;  
if (i % 2 == 0) {  
    i = i + two;  
    // l = i; // not  
} else {  
    k = two * two;  
    l = k;  
}  
jump l; // sink
```