EXERCISE #34

LINTING REVIEW

Write your name and answer the following on a piece of paper

Give an example of a program that a linter might flag as a problem and explain why it would do so.

EXERCISE #34: SOLUTION

LINTING REVIEW

ADMINISTRIVIA AND ANNOUNCEMENTS

Grades still not done ☹

EECS 665 Quiz 4

and

EECS 677 Replacement test conflict

ADMINISTRIVIA AND ANNOUNCEMENTS

ADMINISTRIVIA AND ANNOUNCEMENTS

P3 up tonight

BUG ISOLATION

EECS 677: Software Security Evaluation

Drew Davidson

WHERE WE'RE AT

GRAB-BAG TOPICS!

PREVIOUSLY: SECURE DESIGN

REVIEW LAST LECTURE

DESCRIBED SOME OF THE BEST PRACTICES IN WRITING SECURE SOFTWARE

- The principle of least privilege / privilege separation
- Simplicity
- Open design
- Defense in depth
- Complete mediation
- Fail safe

THIS LECTURE

BUG ISOLATION

Isolating Cause-Effect Chains in program misbehavior

- Why we isolate bugs
- How we isolate bugs

FROM DEFECT TO FAILURE

BUG ISOLATION

THE ANATOMY OF A PROBLEM

Step 1. The programmer creates a *defect* (an error in the code) **We care about this**

Step 2. When executed, the program creates an *infection* (an error in the state)

Step 3. The infection propagates

Step 4. The infection causes a failure / exploit **We see this**

HOW TO FIX A BUG

BUG ISOLATION

UNWINDING A COMPLEX ISSUE REQUIRES CAREFUL CONSIDERATION

Sufficient logging to detect failure / exploit

Sufficient logging to trace back the propagation

Identification of the defect

Insert Simplification

SIMPLIFICATION

BUG ISOLATION

What part of an infection is relevant to the defect?

• Does the problem really depend on 10,000 lines of input?
• Does the failure really require this exact schedule?
• Do we need this sequence of calls?

Experiment-Based Simplification

- For every aspect of the problem, check whether it is relevant for the problem to occur.
- If it is not, remove that aspect from the report or test case

BUG REPORTS

BUG ISOLATION

In practice, even a debug trace might not be available

Consider most open source software – a bug report is likely to only provide a failing case

ACTING ON BUG REPORTS

BUG ISOLATION

ANECDOTE

In 1999 Bugzilla, the bug database for the browser Firefox, listed more than 370 open bugs
Each bug in the database describes a scenario which caused software to fail
these scenarios are not simplified
they may contain a lot of irrelevant information
a lot of the bug reports could be equivalent
Overwhelmed with this work Mozilla developers sent out a call for volunteers
Process the bug reports by producing simplified bug reports
Simplifying means: turning the bug reports into minimal test cases where every part of the input would be significant in reproducing the failure

MOZILLA ANECDOTE: EXAMPLE

BUG ISOLATION

Printing the Following file caused Firefox to crash

```
<td align=left valign=top>
<SELECT NAME="op sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All<OPTION VALUE="Windows 3.1">Windows
3.1<OPTION
VALUE="Windows 95">Windows 95<OPTION VALUE="Windows
98">Windows 98<OPTION VALUE="Windows ME">Windows ME<OPTION
VALUE="Windows 2000">Windows 2000<OPTION VALUE="Windows
NT">Windows NT<OPTION VALUE="Mac System 7">Mac System 7<OPTION
VALUE="Mac
System 7.5">Mac System 7.5<OPTION VALUE="Mac
System 7.6.1">Mac System 7.6.1<OPTION VALUE="Mac System 8.0">Mac
System
8.0<OPTION VALUE="Mac System 8.5">Mac System
8.5<OPTION VALUE="Mac System 8.6">Mac System 8.6<OPTION
VALUE="Mac System
9.x">Mac System 9.x<OPTION VALUE="MacOS X">MacOS
X<OPTION VALUE="Linux">Linux<OPTION VALUE="BSDI">BSDI<OPTION
VALUE="FreeBSD">FreeBSD<OPTION VALUE="NetBSD">NetBSD<OPTION
VALUE="OpenBSD">OpenBSD<OPTION VALUE="AIX">AIX
<OPTION
VALUE="BeOS">BeOS<OPTION VALUE="HP-UX">HP-UX<OPTION
VALUE="IRIX">IRIX<OPTION VALUE="Neutrino">Neutrino<OPTION
VALUE="OpenVMS">
OpenVMS<OPTION VALUE="OS/2">OS/2<OPTION
VALUE="OSF/1">OSF/1<OPTION VALUE="Solaris">Solaris<OPTION
VALUE="SunOS">SunOS<OPTION
VALUE="other">other</SELECT></td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION
VALUE="P2">P2<OPTION
VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION
VALUE="P5">P5</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="bug severity" MULTIPLE SIZE=7>
<OPTION VALUE="blocker">blocker<OPTION
VALUE="critical">critical<OPTION
VALUE="major">major<OPTION
VALUE="normal">normal<OPTION VALUE="minor">minor<OPTION
VALUE="trivial">trivial<OPTION
VALUE="enhancement">enhancement</SELECT>
</tr>
</table>
```

WHY?

CAUSE AND EFFECT

BUG ISOLATION

FREQUENTLY, A SUBSET OF INPUT WILL BE THE CULPRIT AND THE REST IS INCIDENTAL

This circumstance creates both the means and motivation for **minimizing** test cases

View minimization as a binary search (or at least a reduction of search space)

CAUSE AND EFFECT

BUG ISOLATION

```
1 <SELECT NAME="priority" MULTIPLE SIZE=7> F
2 <SELECT NAME="priority" MULTIPLE SIZE=7> P
3 <SELECT NAME="priority" MULTIPLE SIZE=7> P
4 <SELECT NAME="priority" MULTIPLE SIZE=7> P
5 <SELECT NAME="priority" MULTIPLE SIZE=7> F
6 <SELECT NAME="priority" MULTIPLE SIZE=7> F
7 <SELECT NAME="priority" MULTIPLE SIZE=7> P
8 <SELECT NAME="priority" MULTIPLE SIZE=7> P
9 <SELECT NAME="priority" MULTIPLE SIZE=7> P
10 <SELECT NAME="priority" MULTIPLE SIZE=7> F
11 <SELECT NAME="priority" MULTIPLE SIZE=7> P
12 <SELECT NAME="priority" MULTIPLE SIZE=7> P
13 <SELECT NAME="priority" MULTIPLE SIZE=7> P
14 <SELECT NAME="priority" MULTIPLE SIZE=7> P
15 <SELECT NAME="priority" MULTIPLE SIZE=7> P
16 <SELECT NAME="priority" MULTIPLE SIZE=7> F
17 <SELECT NAME="priority" MULTIPLE SIZE=7> F
18 <SELECT NAME="priority" MULTIPLE SIZE=7> F
19 <SELECT NAME="priority" MULTIPLE SIZE=7> P
20 <SELECT NAME="priority" MULTIPLE SIZE=7> P
21 <SELECT NAME="priority" MULTIPLE SIZE=7> P
22 <SELECT NAME="priority" MULTIPLE SIZE=7> P
23 <SELECT NAME="priority" MULTIPLE SIZE=7> P
24 <SELECT NAME="priority" MULTIPLE SIZE=7> P
25 <SELECT NAME="priority" MULTIPLE SIZE=7> P
26 <SELECT NAME="priority" MULTIPLE SIZE=7> F
```

DELTA DEBUGGING

BUG ISOLATION

WHAT PART OF AN INFECTION IS RELEVANT TO THE DEFECT?

It is very tedious (but highly mechanical) to modify and re-test program aspects

Tasty target
for automation!

DELTA DEBUGGING: NEEDS

BUG ISOLATION

A Failing test Case and a Passing test Case

DELTA DEBUGGING: ALGORITHM

BUG ISOLATION

```
def dd(c_pass, c_fail):
    n = 2
    while true:
    delta = listminus(c_fail, c_pass)
    deltas = split(delta, n); offset = 0; j = 0
    while j < n:
        i = (j + offset) % n
        next_c_pass = listunion(c_pass, deltas[i])
        next_c_fail = listminus(c_fail, deltas[i])
        if test(next_c_fail) == FAIL and n == 2:
            c_fail = next_c_fail; n = 2; offset = 0; break
        elif test(next_c_fail) == PASS:
            c_pass = next_c_fail; n = 2; offset = 0; break
        elif test(next_c_pass) == FAIL:
            c_fail = next_c_pass; n = 2; offset = 0; break
        elif test(next_c_fail) == FAIL:
            c_fail = next_c_fail; n = max(n - 1, 2); offset = i; break
        elif test(next_c_pass) == PASS:
            c_pass = next_c_pass; n = max(n - 1, 2); offset = i; break
        else:
            j = j + 1
    if j >= n:
        if n >= len(delta):
            return (delta, c_pass, c_fail)
        else:
            n = min(len(delta), n * 2)
```

DELTA DEBUGGING: APPLICATIONS

BUG ISOLATION

IT'S NOT JUST FOR MANIPULATING INPUT!

Consider determining bugs caused by…

- Code changes
- Thread interleavings

COOPERATIVE BUG ISOLATION

BUG ISOLATION

Fault Localization Is Expensive!

Gathering sufficient telemetry slows down programs

Much of the logging won’t be useful in the end

Key Idea

Statistically distribute logging across the userbase

Good news! Works best in the circumstances where it is most needed

COOPERATIVE BUG ISOLATION

BUG ISOLATION

Basic Scheme

Each user records 1% of everything

Adapts the sparse sampling scheme by Arnold and Ryder

How to Sample?

```
check (p != NULL);
p = p->next;
check (i < max);
total += sizes[i];
```

One idea: randomize ...

```
if (rand(100)== 0){ check (p != NULL); }
p = p->next;
if (rand(100) == 0){ check (i < max); }
total += sizes[i];
```

... but rand(100) is super expensive!

COOPERATIVE BUG ISOLATION

BUG ISOLATION

How to Sample?

```
check (p != NULL);
p = p->next;
check (i < max);
total += sizes[i];
```

One idea: randomize …

```
if (rand(100)== 0){ check (p != NULL); }
p = p->next;
if (rand(100) == 0){ check (i < max); }
total += sizes[i];
```

… but rand(100) is super expensive!

Another idea: global counter

```
if (k++ % 100 == 0){ check (p != NULL); }
p = p->next;
if (k++ % 100 == 0){ check (i < max); }
total += sizes[i];
```

You’ll never get the second check!

COOPERATIVE BUG ISOLATION

BUG ISOLATION

How to Sample?

```
check (p != NULL);
p = p->next;
check (i < max);
total += sizes[i];
```

One idea: randomize …

```
if (rand(100)== 0){ check (p != NULL); }
p = p->next;
if (rand(100) == 0){ check (i < max); }
total += sizes[i];
```

… but rand(100) is super expensive!

CBI’s working solution

- Use a randomized global countdown
- Restore the countdown by sampling from a geometric distribution

Another idea: global counter

```
if (k++ % 100 == 0){ check (p != NULL); }
p = p->next;
if (k++ % 100 == 0){ check (i < max); }
total += sizes[i];
```

You’ll never get the second check!

COOPERATIVE BUG ISOLATION

BUG ISOLATION

CBI'S SAMPLING METHOD

- Use a randomized global countdown
- Restore the countdown by sampling from a geometric distribution

Benefits

- Doesn't use clock interrupt
- Isn't periodic
- Deciding to check is relatively quick

COOPERATIVE BUG ISOLATION

BUG ISOLATION

Real CBI is slightly more complex

Smart(er) about what points to instrument

- Essentially finds acyclic regions of the control flow and instruments intelligently
- Clones regions of code with a “fast” variant and a “slow” variant

A number of optimizations exist to make countdown lookup faster

- e.g. Caching a global variable in local function such that it might be better optimized without interprocedural analysis

WRAP-UP

BUG ISOLATION

IMPORTANCE OF SIMPLIFICATION IN FIXING/SECURING PROGRAMS

Methods include collaborative bug isolation / delta debugging