

---

# **Random Testing: Not Just for Unit Tests**

---

# Random Testing: How Good is It?

- Random testing is clearly sometimes both easy and effective
  - For what are essentially unit tests of a small piece of code
  - Can it work on larger pieces of software?

***YES!***

---

# Random Testing in the Real World

- Random testing (usually called “fuzzing” in this context) is highly effective
  - Mozilla and Google use random testing to detect critical security flaws in JavaScript engines and browsers
    - Search for Google’s “ClusterFuzz”
  - Extremely effective for finding bugs in C compilers, including over 400 in GCC and LLVM
    - Search for the Csmith compiler testing project
  - Used to find bugs in Apache commons, Java core libraries, other widely used code
  - At NASA, random testing is used to test file systems for missions with costs well over \$100M

---

# Industrial Strength Random Testing

- No longer a quick afternoon's effort
  - Engineering random testers for compilers, interpreters, large libraries is a major undertaking
    - Requires testing expertise and domain expertise
  - Have to consider new issues such as:
    - How long should tests be?
    - How much to test each API call or function?
    - How to identify different bugs when an overnight testing run produces 5,000+ failing tests?
      - But there are only 20 different bugs
    - Can developers debug test cases consisting of thousands of random operations, mostly irrelevant?

# A Very Bad Test Case

```

/cygdrive/c/Documents and Settings/Alex/Desktop/cs362class/dominion
tryItOut("with({constructor: (__iterator__ = eval(\"let prototype = <><x><y/></x></>\", this)))let (__it
erator__ = this.__defineGetter__(\"__iterator__\", function () { yield; } )) { var __parent__ = *, __count
__; }");
tryItOut("windowlet (constructor) { ; }");
tryItOut("let(__noSuchMethod__ = <><x><y/></x></> { 0});
tryItOut("__proto__.__noSuchMethod__ = __parent__");
tryItOut("if(()) yield __count__; else if (__parent__ = null) [1,2,3,4].map else {yield; }");
tryItOut("/for..in*/M:for(var __parent__ in ((({__count__: [] })).unwatch(\"constructor\"))({prop getter
r: (new Function(\"yield __count__.throw(*:*)\"; }))){{with({prop: {}}.__defineSetter__(\"__parent__\",
function prototype(prop, prototype) { } })}} }");
tryItOut("do return eval(\"(-1)\", window); while((((__iterator__ getter: (new Function(\"continue M;\")
})) && 0));");
tryItOut("return ;");
tryItOut("/for..in*/L:for(let [__iterator__, prop] = (__noSuchMethod__.__parent__ = __iterator__) in fals
e < window) {L: {<><x><y/></x></> } }");
tryItOut("/for..in*/M:for(const [prototype, prototype] = 1e-81 in __count__.throw(<><x><y/></x></>));");
tryItOut("M:do return 4;var __parent__;constructor = undefined; while((++break ; ) && 0);");
tryItOut("/for..in*/for(var [ <><x><y/></x></> , __noSuchMethod__ ] = * in this) let __parent__\n3/0");
tryItOut("M:with({__count__: false() * (__parent__.__count__)})continue M;");
tryItOut("L:do yield 1.2e3; while((__count__.prototype|=033 <= \fprop) && 0);");
tryItOut("#1#\n");
tryItOut("/for..in*//* noageckoex bug 349964 */L: for each(const __noSuchMethod__ in [1]) ");
tryItOut("const __parent__");
tryItOut("let (__proto__, prop = 3.141592653589793) { return; }");
tryItOut("with({}) return false == prop;");
tryItOut("for(let y in []);");
tryItOut("yield\nreturn __count__");
tryItOut("L: *");
tryItOut("yield;");
tryItOut("let(__count__ = 033) { with({}) for(let y in [5,6,7,8]) return;const __proto__");
tryItOut("/for..in*/for(var __proto__ in (((function(id) { return id } ).call)(1e4)))<><x><y/></x></> }"
);
tryItOut("/for..in*/for(const ((function ( )*).call).call in ((*:*)(false)))var __noSuchMethod__ = *:*\n
");
tryItOut("with({}) for(let y in []);");
tryItOut("for(let y in []);");
tryItOut("/for..in*/for(const __noSuchMethod__ in (((null).watch)(<><x><y/></x></>))<><x><y/></x></>");
tryItOut("/for..in*/L:for(var [__noSuchMethod__, __iterator__] = [11,12,13,14].map in 1.2e3) for(let y in
[]);");
tryItOut("do while((Object(window)) && 0)null; while((__proto__|=) && 0);");
tryItOut("let (__noSuchMethod__ = __count__+=new prop = 4.(), __count__ = prop-={}.valueOf(\"number\")) {
with({__parent__: *:*)let constructor = window; }");
tryItOut("let(prop = __iterator__, __iterator__ = undefined) { return window.prop\n@foo;");
tryItOut("with({}) ");
tryItOut("if(*:*) {yield false; } else if (1.2e3) yield; else {return; }");
tryItOut("throw StopIteration;");
tryItOut("let (__parent__) { [1,2,3,4].map }yield;");
tc84953.full lines 1-37/1000 2%
```

tryItOut("do; while(\_\_parent\_\_)\_\_proto\_\_=\_\_parent\_\_")  
tryItOut("while(Error(this)){}")

---

# Test Case Minimization

- Solution: automatic *minimization* of test cases as they are generated
- Minimized test case: subset of original sequence of operations such that
  - Test case still fails
  - Removing any one operation makes the test case successful
- Typical improvement: order of magnitude or greater reduction in length of a test case
  - Highly effective technique, essential for quick debugging

---

# Test Case Minimization

- Based on Zeller's *delta-debugging* tools
  - Automated debugging state-of-the-art
  - Set of Python scripts easily modified to automatically minimize tests in different settings
  - Requires that you be able to
    - Play back test cases and determine success or failure automatically
    - Define the subsets of a test case – provide a test case decomposition

**REMEMBER DELTA DEBUGGING: IT CAN  
SAVE YOU ENDLESS HOURS OF EFFORT**

# Test Case Minimization

- Based on a clever modification of a “binary search” strategy

