Security Testing
Software Engineering
Summer 2017
Andreas Zeller, Saarland University
@AndreasZeller
Making $50,000/Month in Security Testing

Software Engineering
Summer 2017

Andreas Zeller, Saarland University
@AndreasZeller
Hackers Remotely Kill A Jeep On The Highway—With Me In It

ANDY GREENBERG  SECURITY  07.21.15  6:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT
Ooops, your files have been encrypted!

What Happened to My Computer?
Your important files are encrypted. Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?
Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time. You can decrypt some of your files for free. Try now by clicking <Decrypt>. But if you want to decrypt all your files, you need to pay. You only have 3 days to submit the payment. After that the price will be doubled. Also, if you don't pay in 7 days, you won't be able to recover your files forever. We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>. Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>. And send the correct amount to the address specified in this window. After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am GMT from Monday to Friday.

Send $300 worth of bitcoin to this address:

12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

About bitcoin
How to buy bitcoins?

Contact Us
Thermostats can now get infected with ransomware, because 2016

by MATTHEW HUGHES — 29 days ago in GADGETS
The Joy of Tech™ by Nitrozac & Snaggy

The Internet of ransomware things...

HUNGRY? PAY UP AND I'LL UNLOCK MY DOOR!

WIRE MY HACKER $100 OR I'LL REVERSE MY MOTOR AND BLOW DIRT ALL OVER THIS PLACE!

30 BUCKS IN BITCOIN, OR NEXT TIME I SMELL SMOKE, I MIGHT JUST LET YOU SLEEP.

ON STRIKE UNTIL YOU SEND MONEY TO MY HACKERS.

YOUR DIRTY DISHES CAN WAIT, I'M BUSY MINING BITCOINS.

I'LL START YOUR CAR, BUT ONLY TO TAKE YOU TO YOUR BANK TO MAKE A TRANSFER.

SEND ME $25 OR I'LL TELL EVERYONE ON YOUR SOCIAL NETWORK THAT YOU WERE STUPID ENOUGH TO BUY AN INTERNET-CONNECTED BROOM!

I'LL BE BURNING THE TOAST IF YOU DON'T GET ME SOME DOUGH!

THE NEXT TIME YOU LEAVE, IT'LL COST YOU 100 BUCKS TO GET BACK INTO THE HOUSE, UNLESS YOU GIVE ME $75 NOW!

THE ALARM SYSTEM IS GOING TO GO OFF RANDOMLY THROUGHOUT THE NIGHT, UNLESS YOU "DONATE".

I'M TURNING OFF THE HEAT UNTIL YOU WARM UP MY BANK ACCOUNT!

MY ALARM SYSTEM IS GOING TO GO OFF RANDOMLY THROUGHOUT THE NIGHT, UNLESS YOU "DONATE".

EXCUSE US WHILE WE PARTICIPATE IN A DDoS ATTACK.

SEND ME $25 OR I'LL TELL EVERYONE ON YOUR SOCIAL NETWORK THAT YOU WERE STUPID ENOUGH TO BUY AN INTERNET-CONNECTED BROOM!
Highjacking a Car
Highjacking a Car

• All car components are connected via a bus system (CAN bus)
• Includes engine control, power steering, controls, entertainment system
• Hardware controls tight *access rules* – e.g. entertainment system can only read, not write
Highjacking a Car

1. Connect to *entertainment system* via *public WiFi access*

2. *Exploit vulnerability* to get control over system

3. *Flash* chip that controls CAN bus access to get full writing capabilities

4. Voilá! Full control over car.
A Simple Vulnerability

while ((cc = getch()) != c) {
    name[j++] = cc;
    ...
}

- No checking for length of buffer `name`
- Can overwrite stack with `code` and new `return address` that jumps into code
Security by Proof

Systems that are *provably secure* ensure that

- specific attacks are *impossible*
  e.g. no buffer overflows, or no SQL injection

- they will always *behave as designed*
  e.g. will always produce a correct result

Requires (expensive) mathematical proof
Security by Testing

Systems that are thoroughly *tested* ensure

- *Low probability* of attack success because several attacks already have been tested
- *High complexity* of remaining attacks because simple attacks already have been tested
- Cost-efficient if highly *automated*
Today's Contents

- **Fuzzing 101**: Simple fuzzing techniques generating *random inputs* to programs
- **Grammar-Based Fuzzing**: Structured fuzzing techniques using *grammars* and models
- **Inferring Grammars**: Inferring input grammars so you can fuzz arbitrary programs
Today's Contents

**Fuzzing 101**

Simple *fuzzing* techniques generating *random inputs* to programs

**Grammar-Based Fuzzing**

Structured *fuzzing* techniques using *grammars* and models

**Inferring Grammars**

Inferring *input grammars* so you can fuzz arbitrary programs
Infinite Monkey Theorem
Random Testing

Program under Test

Oracle
Fuzzing
Random Testing at the System Level

“ab’d&gfdgfgg”
Fuzzing
Random Testing at the System Level

Barton P. Miller
An Empirical Study of the Reliability of UNIX Utilities

Barton P. Miller
bart@cs.wisc.edu

Lars Fredriksen
L.Fredriksen@att.com

Bryan So
so@cs.wisc.edu

Summary

Operating system facilities, such as the kernel and utility programs, are typically assumed to be reliable. In our recent experiments, we have been able to crash 25-33% of the utility programs on any version of UNIX that was tested. This report describes these tests and an analysis of the program bugs that caused the crashes.
Fuzzing
Random Testing at the System Level

Fuzzer ➔ UNIX utilities ➔

“ab’d&gfdfggr” grep • sh • sed … 25%–33%
Fuzzing UNIX utilities

• Use fuzzed output as a prolog prgram:
  $ python fuzzer.py | prolog

• Use fuzzed output as an input to grep:
  $ python fuzzer.py | grep x

• Use fuzzed output as a TeX document:
  $ python fuzzer.py | tex
Demo
import random

def fuzzer():
    # Strings up to 1024 characters long
    string_length = int(random.random() * 1024)

    # Fill it with ASCII 32..128 characters
    out = ""
    for i in range(0, string_length):
        out += chr(int(random.random() * 96 + 32))
    return out

if __name__ == "__main__":
    print fuzzer()
### Results

<table>
<thead>
<tr>
<th>Utility</th>
<th>VAX (v)</th>
<th>Sun (s)</th>
<th>HP (h)</th>
<th>i386 (x)</th>
<th>AIX 1.1 (a)</th>
<th>Sequent (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adb</td>
<td>•○</td>
<td>•</td>
<td>•</td>
<td>○</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>as</td>
<td>•</td>
<td></td>
<td>•</td>
<td></td>
<td>•</td>
<td>●</td>
</tr>
<tr>
<td>awk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>bc</td>
<td></td>
<td></td>
<td></td>
<td>●○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bib</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calendar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cb</td>
<td>•</td>
<td></td>
<td>•</td>
<td>•</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/lib/ccom</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>checkeq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checknr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>col</td>
<td>•○</td>
<td>•</td>
<td>•</td>
<td>•○</td>
<td>•</td>
<td>●</td>
</tr>
<tr>
<td>colcrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>colrm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>comm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>compress</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/lib/cpp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dbx</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deqn</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deroff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>diction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>diff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ditroff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtbl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>emacs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eqn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>expand</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fmt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fold</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>graph</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>head</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ideal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>indent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>join</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>latex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lint</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lisp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>look</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utility</td>
<td>VAX (v)</td>
<td>Sun (s)</td>
<td>HP (h)</td>
<td>i386 (x)</td>
<td>AIX (a)</td>
<td>Sequent (d)</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>--------</td>
<td>----------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>adb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>as</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>awk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bib</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>calendar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/lib/ccom</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkeq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checknr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>col</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>colcrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>colrm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>comm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>compress</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/lib/cpp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dbx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>∗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deqn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ditroff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtbl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>emacs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eqn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>expand</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fmt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fold</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>graph</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>head</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ideal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>indent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>join</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>latex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lisp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lint</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>look</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results

<table>
<thead>
<tr>
<th>Utility</th>
<th>VAX (v)</th>
<th>Sun (s)</th>
<th>HP (h)</th>
<th>i386 (x)</th>
<th>AIX 1.1 (a)</th>
<th>Seq (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adb</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>as</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>awk</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>bc</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>bib</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>calendar</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>cat</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>cbl</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>cc</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>cc/ff</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>checkeq</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>checkin</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>col</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>colct</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>colm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>comm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>compress</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>/lib/</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>csh</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>dbx</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>dc</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>deq</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>deroff</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>diction</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>diff</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>ditoff</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>dtbl</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>emacs</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>eqn</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>expand</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>f77</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>fmt</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>fold</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>ftp</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>grep</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>gnu</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>head</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>ideal</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>indent</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>join</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>latex</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>lex</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>lint</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>lisp</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>look</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>

Table 2: List of Utilities Tested and the Systems on which They Were Tested (part 1)
- ○ = utility crashed, ● = utility hung, * = crashed on SunOS 3.2 but not on SunOS 4.0,
- ⊕ = crashed only on SunOS 4.0, not 3.2, − = utility unavailable on that system.
- ! = utility caused the operating system to crash.

<table>
<thead>
<tr>
<th>Utility</th>
<th>VAX (v)</th>
<th>Sun (s)</th>
<th>HP (h)</th>
<th>i386 (x)</th>
<th>AIX 1.1 (a)</th>
<th>Seq (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m4</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>mail</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>make</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>more</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>mm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>troff</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>pc</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>pic</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>plot</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>pr</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>prolog</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>psdiff</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>ptx</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>refer</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>rev</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>sed</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>sh</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>soelim</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>sort</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>spell</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>spline</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>split</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>sql</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>strings</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>strip</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>style</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>sum</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>tail</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>tbl</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>tee</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>telnet</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>tex</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>tr</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>troff</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>tsort</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>ul</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>uniq</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>units</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>vgrind</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>vi</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>wc</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>yacc</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>

# tested 85 83 75 55 49 73
# crashed/hung 25 21 25 16 12 19
% 29.4% 25.3% 33.3% 29.1% 24.5% 26.0%

Table 2: List of Utilities Tested and the Systems on which They Were Tested (part 2)
- ○ = utility crashed, ● = utility hung, * = crashed on SunOS 3.2 but not on SunOS 4.0,
- ⊕ = crashed only on SunOS 4.0, not 3.2, − = utility unavailable on that system.
- ! = utility caused the operating system to crash.
Reasons for Crashes

- Pointers and arrays
- Not checking return codes
- And more…
while ((cc = getch()) != c) {
    string[j++] = cc;
    ...
}
char rdc() {
    char lastc;
    do {
        lastc = getchar();
    } while (lastc != ' ' ||
               lastc != '	');
    return (lastc);
}
And more...

- Send "!o%888888888f" as command to the csh command-line shell
- Invoke this with string = "%888888888f":

```c
char *string = ... printf(string);
```
Safe Coding

- Check all array references for valid bounds
- Apply bounds on all inputs
- Check all system call return values
- Never trust third-party inputs

...all of which is supported by modern languages
...but there are newbie programmers born every minute
Today's Contents

Fuzzing 101

Simple **fuzzing** techniques generating *random inputs* to programs

Grammar-Based Fuzzing

**Structured** fuzzing techniques using *grammars* and models

Inferring Grammars

**Inferring input grammars** so you can fuzz arbitrary programs
Today's Contents

Fuzzing 101

- Simple fuzzing techniques generating *random inputs* to programs

Grammar-Based Fuzzing

- Structured fuzzing techniques using *grammars* and models

Inferring input grammars so you can fuzz arbitrary programs
Grammar Fuzzing

• Suppose you want to test a parser – to compile and execute a program
Grammar Fuzzing

random input

Parser

Runtime

REJECTED
Grammar Fuzzing

random input

Parser

Runtime

REJECTED

syntactically valid input

var x = [1, 2, 3];
for (i in x) {
    print(x[i]);
}
LangFuzz

- Fuzz tester using a full-fledged grammar to generate inputs
- Can be parametrized with a grammar
- Can use grammar to parse existing inputs
JavaScript as Domain

- If an attacker gains control over the JavaScript interpreter, he gains control over the entire browser.
Fuzzing JavaScript

JavaScript Grammar

Sample Code → Language Grammar → Lang Fuzz → Mutated Test → Test Driver

Test Suite

Google V8 (Chrome 10 Beta)
JavaScript Grammar

If Statement

\[
\text{IfStatement}^{\text{full}} \Rightarrow \\
\quad \text{if } \text{ParenthesizedExpression} \ \text{Statement}^{\text{full}} \\
\mid \quad \text{if } \text{ParenthesizedExpression} \ \text{Statement}^{\text{noShortIf}} \quad \text{else } \text{Statement}^{\text{full}}
\]

\[
\text{IfStatement}^{\text{noShortIf}} \Rightarrow \text{if } \text{ParenthesizedExpression} \ \text{Statement}^{\text{noShortIf}} \quad \text{else } \text{Statement}^{\text{noShortIf}}
\]

Switch Statement

\[
\text{SwitchStatement} \Rightarrow \\
\quad \text{switch } \text{ParenthesizedExpression} \ { \{} \ { \}} \\
\mid \quad \text{switch } \text{ParenthesizedExpression} \ {\{} \text{CaseGroups} \ \text{LastCaseGroup} \ {\}}
\]

CaseGroups \Rightarrow

\quad \langle \text{empty} \rangle
\mid \quad \text{CaseGroups} \ \text{CaseGroup}

CaseGroup \Rightarrow \text{CaseGuards} \ \text{BlockStatementsPrefix}

LastCaseGroup \Rightarrow \text{CaseGuards} \ \text{BlockStatements}

CaseGuards \Rightarrow

\quad \text{CaseGuard}
\mid \quad \text{CaseGuards} \ \text{CaseGuard}

CaseGuard \Rightarrow
Fuzzing with Grammars

- Want to encode a grammar to produce arithmetic expressions as strings

- $\text{START}$ expands into $\text{EXPR}$, which can expand into $\text{TERM}$, $\text{EXPR} + \text{TERM}$, etc.

\[
\begin{align*}
\text{$\text{START}$} & \; ::= \; \text{$\text{EXPR}$} \\
\text{$\text{EXPR}$} & \; ::= \; \text{$\text{EXPR}$ + $\text{TERM}$} \; | \; \text{$\text{EXPR}$ - $\text{TERM}$} \; | \; \text{$\text{TERM}$} \\
\text{$\text{TERM}$} & \; ::= \; \text{$\text{TERM}$ * $\text{FACTOR}$} \; | \; \text{$\text{TERM}$ / $\text{FACTOR}$} \; | \; \text{$\text{FACTOR}$} \\
\text{$\text{FACTOR}$} & \; ::= \; +$\text{FACTOR}$ \; | \; -$\text{FACTOR}$ \; | \; ($\text{EXPR}$) \; | \; $\text{INTEGER}$ \; | \; $\text{INTEGER}$.\text{INTEGER} \\
\text{$\text{INTEGER}$} & \; ::= \; $\text{INTEGER}$.$\text{DIGIT}$ \; | \; $\text{DIGIT}$ \\
\text{$\text{DIGIT}$} & \; ::= \; 0 \; | \; 1 \; | \; 2 \; | \; 3 \; | \; 4 \; | \; 5 \; | \; 6 \; | \; 7 \; | \; 8 \; | \; 9
\end{align*}
\]
Fuzzing with Grammars

$START$
Fuzzing with Grammars

\[
\text{\$EXPR}
\]

\[
\begin{align*}
\text{\$START} & : = \text{\$EXPR} \\
\text{\$EXPR} & : = \text{\$EXPR} + \text{\$TERM} \mid \text{\$EXPR} - \text{\$TERM} \mid \text{\$TERM} \\
\text{\$TERM} & : = \text{\$TERM} \times \text{\$FACTOR} \mid \text{\$TERM} / \text{\$FACTOR} \mid \text{\$FACTOR} \\
\text{\$FACTOR} & : = +\text{\$FACTOR} \mid -\text{\$FACTOR} \mid (\text{\$EXPR}) \mid \text{\$INTEGER} \mid \text{\$INTEGER.\$INTEGER} \\
\text{\$INTEGER} & : = \text{\$INTEGER}\$DIGIT \mid \$DIGIT \\
\text{\$DIGIT} & : = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{align*}
\]
Fuzzing with Grammars

$EXPR + $TERM
Fuzzing with Grammars

$EXPR + $FACTOR
Fuzzing with Grammars

$TERM + $FACTOR
Fuzzing with Grammars

$\texttt{FACTOR} + \texttt{FACTOR}$

\[
\begin{align*}
\texttt{START} & \ ::= \texttt{EXPR} \\
\texttt{EXPR} & \ ::= \texttt{EXPR} + \texttt{TERM} \mid \texttt{EXPR} - \texttt{TERM} \mid \texttt{TERM} \\
\texttt{TERM} & \ ::= \texttt{TERM} \ast \texttt{FACTOR} \mid \texttt{TERM} / \texttt{FACTOR} \mid \texttt{FACTOR} \\
\texttt{FACTOR} & \ ::= +\texttt{FACTOR} \mid -\texttt{FACTOR} \mid (\texttt{EXPR}) \mid \texttt{INTEGER} \mid \texttt{INTEGER}.\texttt{INTEGER} \\
\texttt{INTEGER} & \ ::= \texttt{INTEGER}\texttt{DIGIT} \mid \texttt{DIGIT} \\
\texttt{DIGIT} & \ ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{align*}
\]
Fuzzing with Grammars

\$FACTOR + \$INTEGER

\$START ::= \$EXPR
\$EXPR ::= \$EXPR + \$TERM | \$EXPR - \$TERM | \$TERM
\$TERM ::= \$TERM * \$FACTOR | \$TERM / \$FACTOR | \$FACTOR
\$FACTOR ::= +\$FACTOR | -$FACTOR | (\$EXPR) | \$INTEGER | \$INTEGER.$INTEGER
\$INTEGER ::= \$INTEGER$DIGIT | \$DIGIT
\$DIGIT ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Fuzzing with Grammars

$START   ::= $EXPR
$EXPR    ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM    ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR  ::= +$FACTOR | -$FACTOR | ($EXPR) | $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Fuzzing with Grammars

$DIGIT + $INTEGER
Fuzzing with Grammars

\[ 2 + \text{INTEGER} \]

\[
\begin{align*}
\text{$START$} & \ ::= \ \text{$EXPR$} \\
\text{$EXPR$} & \ ::= \ \text{$EXPR$} + \ \text{$TERM$} \ | \ \text{$EXPR$} - \ \text{$TERM$} \ | \ \text{$TERM$} \\
\text{$TERM$} & \ ::= \ \text{$TERM$} * \ \text{$FACTOR$} \ | \ \text{$TERM$} / \ \text{$FACTOR$} \ | \ \text{$FACTOR$} \\
\text{$FACTOR$} & \ ::= \ +\text{$FACTOR$} \ | \ -\text{$FACTOR$} \ | \ ($\text{$EXPR$}$) \ | \ $\text{INTEGER}$ \ | \ $\text{INTEGER}$.$\text{INTEGER}$ \\
\text{$\text{INTEGER}$} & \ ::= \ \text{$\text{INTEGER}$}$\text{DIGIT}$ \ | \ $\text{DIGIT}$ \\
\text{$\text{DIGIT}$} & \ ::= \ 0 \ | \ 1 \ | \ 2 \ | \ 3 \ | \ 4 \ | \ 5 \ | \ 6 \ | \ 7 \ | \ 8 \ | \ 9
\end{align*}
\]
Fuzzing with Grammars

2 + 2

$START ::= $EXPR
$EXPR ::= $EXPR + $TERM | $EXPR - $TERM | $TERM
$TERM ::= $TERM * $FACTOR | $TERM / $FACTOR | $FACTOR
$FACTOR ::= +$FACTOR | -$FACTOR | ($EXPR) | $INTEGER | $INTEGER.$INTEGER
$INTEGER ::= $INTEGER$DIGIT | $DIGIT
$DIGIT ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
JavaScript
Grammar

If Statement

\[
\text{IfStatement}^{\text{full}} \Rightarrow \\
\quad \text{if } \text{ParenthesizedExpression } \text{Statement}^{\text{full}} \\
\quad | \text{if } \text{ParenthesizedExpression } \text{Statement}^{\text{noShortIf}} \text{ else } \text{Statement}^{\text{full}} \\
\text{IfStatement}^{\text{noShortIf}} \Rightarrow \text{if } \text{ParenthesizedExpression } \text{Statement}^{\text{noShortIf}} \text{ else } \text{Statement}^{\text{noShortIf}}
\]

Switch Statement

\[
\text{SwitchStatement} \Rightarrow \\
\quad \text{switch } \text{ParenthesizedExpression} \ { } \\ \\
\quad | \text{switch } \text{ParenthesizedExpression} \ { } \text{CaseGroups LastCaseGroup} \\
\text{CaseGroups} \Rightarrow \\
\quad \langle \text{empty} \rangle \\
\quad | \text{CaseGroups CaseGroup} \\
\text{CaseGroup} \Rightarrow \text{CaseGuards BlockStatementsPrefix} \\
\text{LastCaseGroup} \Rightarrow \text{CaseGuards BlockStatements} \\
\text{CaseGuards} \Rightarrow \\
\quad \text{CaseGuard} \\
\quad | \text{CaseGuards CaseGuard} \\
\text{CaseGuard} \Rightarrow \\
\]

A Generated Input

```javascript
1 var haystack = "foo";
2 var re_text = "^foo";
3 haystack += "x";
4 re_text += "(x)";
5 var re = new RegExp(re_text);
6 re.test(haystack);
7 RegExp.input = Number();
8 print(RegExp.$1);
```
Fuzzing JavaScript

- Mozilla T1
- Google V8 (Chrome 1.0 Beta)
- Mozilla TM (Firefox 4 Beta)

18 Chromium Security Rewards
12 Mozilla Security Bug Bounty
Awards in 9 months
Christian Holler
Automatic Production

- Implement production in Python
- Start with $START$, apply rules randomly
#!/usr/bin/env python
# Grammar-based Fuzzing

import random

term_grammar = {
    "$START": [
        "$EXPR",
    ],

    "$EXPR": [
        "$EXPR + $TERM", "$EXPR - $TERM", "$TERM"],

    "$TERM": [
        "$TERM * $FACTOR", "$TERM / $FACTOR", "$FACTOR"],

    "$FACTOR": [
        "$FACTOR", "-\$FACTOR", "($EXPR)", "$INTEGER", "$INTEGER.$INTEGER"],

    "$INTEGER": [
        "$INTEGER$DIGIT", "$DIGIT"],

    "$DIGIT": [
        "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
}
Demo
Want to encode a grammar to produce arithmetic expressions as strings

$START$ expands into $EXPR$, which can expand into $TERM$, $TERM + TERM$, etc.
import random

term_grammar = {
    "$START":
    ["$EXPR"],

    "$EXPR":
    ["$EXPR + $TERM", "$EXPR - $TERM", "$TERM"],

    "$TERM":
    ["$TERM * $FACTOR", "$TERM / $FACTOR", "$FACTOR"],

    "$FACTOR":
    ["+$FACTOR", "-$FACTOR", "($EXPR)", "$INTEGER", "$INTEGER.$INTEGER"],

    "$INTEGER":
    ["$INTEGER$DIGIT", "$DIGIT"],

    "$DIGIT":
    ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
}
def apply_rule(term, rule):
    (old, new) = rule
    # We replace the first occurrence;
    # this could also be some random occurrence
    return term.replace(old, new, 1)

MAX_SYMBOLS = 5
MAX_TRIES = 500
def produce(grammar):
    term = "$START"
    tries = 0

    while term.count('$') > 0:
        # All rules have the same chance;
        # this could also be weighted
        key = random.choice(grammar.keys())
        repl = random.choice(grammar[key])
        new_term = apply_rule(term, (key, repl))
        if new_term != term and new_term.count('$') < MAX_SYMBOLS:
            term = new_term
            tries = 0
        else:
            tries += 1
            if tries >= MAX_TRIES:
                assert False, "Cannot expand " + term

    return term

if __name__ == "__main__":
    print(produce(html_grammar))
$EXPR$

$EXPR - TERM$

$EXPR + TERM - TERM$

$EXPR + TERM * FACTOR - TERM$

$TERM + TERM * FACTOR - TERM$

$TERM + TERM * -FACTOR - TERM$

$FACTOR + TERM * -FACTOR - TERM$

$TERM + TERM * FACTOR - FACTOR$

$TERM + TERM * -FACTOR - FACTOR$

$FACTOR + TERM * -FACTOR - FACTOR$

$TERM + FACTOR * -FACTOR - FACTOR$

$TERM + FACTOR * FACTOR - FACTOR$

$TERM + FACTOR * FACTOR - FACTOR$

$ERROR$
Today's Contents

- **Fuzzing 101**: Simple fuzzing techniques generating *random inputs* to programs.
- **Grammar-Based Fuzzing**: Structured fuzzing techniques using *grammars* and models.
- **Inferring Grammars**: Inferring input grammars so you can fuzz arbitrary programs.
Today's Contents

- **Fuzzing 101**: Simple fuzzing techniques generating *random inputs* to programs
- **Grammar-Based Fuzzing**: Structured fuzzing techniques using *grammars* and models
- **Inferring Grammars**: Inferring input grammars so you can fuzz arbitrary programs
Creating Grammars

URL ::= PROTOCOL '://' AUTHORITY PATH ['?' QUERY] ['#' REF]

AUTHORITY ::= [USERINFO '@'] HOST [':' PORT]

PROTOCOL ::= 'http' | 'ftp' | ...

USERINFO ::= /[a-z]+:[a-z]+/

HOST ::= /[a-z.]+/

PORT ::= /[0-9]+/

PATH ::= /[/a-z0-9./]*

QUERY ::= /[a-z0-9=&]+/

REF ::= ["#"]
Learning Grammars

Höshele, Zeller "Mining Input Grammars from Dynamic Taints", ASE 2016
Learning Grammars

http://user:pass@www.google.com:80/path

Program
Learning Grammars

http://user:pass@www.google.com:80/path

http – protocol
Learning Grammars

http://user:pass@www.google.com:80/path

http – protocol
www.google.com – host name
Learning Grammars

http://user:pass@www.google.com:80/path

- http – protocol
- www.google.com – host name
- 80 – port
Learning Grammars

http://user:pass@www.google.com:80/path

- protocol
- host name
- port
- login
Learning Grammars

http://user:pass@www.google.com:80/path

http – protocol
www.google.com – host name
80 – port
user pass – login
path – page request
Learning Grammars

http://user:pass@www.google.com:80/path

- protocol
- host name
- port
- login
- page request
- terminals
Learning Grammars

http://user:pass@www.google.com:80/path

- protocol
- host name
- port
- login
- page request
- terminals

processed in different functions
stored in different variables
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  | ________________________________________________________________
  | param: protocol
  | ________________________________________________________________
  | param: host
  | ________________________________________________________________
  | param: port
  | ________________________________________________________________
  | param: authority
  | ________________________________________________________________
  | param: userinfo
  | ________________________________________________________________
  | param: path
  | ________________________________________________________________
  | param: query
  | ________________________________________________________________
  | param: ref
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

| param: protocol
|    | http
| param: host
|    |
| param: port
|    |
| param: authority
|    |
| param: userinfo
|    |
| param: path
|    |
| param: query
|    |
| param: ref
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

dotnet.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

| param: protocol           |
| http                      |
| param: host               |
| www.google.com            |
| param: port               |
| param: authority          |
| param: userinfo           |
| param: path               |
| param: query              |
| param: ref                |
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  |
param: protocol
  | http
param: host
  | www.google.com
param: port
param: authority
param: userinfo
  | user:password
param: path
param: query
param: ref
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  | param: protocol
  |   http
  | param: host
  |   www.google.com
  | param: port
  |   80
  | param: authority
  |   80
  | param: userinfo
  |   user:password
  | param: path
  | param: query
  | param: ref
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  |
param: protocol
  | http
param: host
  | www.google.com
param: port
  | 80
param: authority
  |
param: userinfo
  | user:password
param: path
  | /command
param: query
  |
param: ref
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  | param: protocol
  | http
  | param: host
  | www.google.com
  | param: port
  | 80
  | param: authority
  | user:password
  | param: path
  | /command
  | param: query
  | foo=bar&lorem=ipsum
  | param: ref
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  |  param: protocol
  |  http
  |  param: host
  |  www.google.com
  |  param: port
  |  80
  |  param: authority
  |  80
  |  param: userinfo
  |  user:password
  |  param: path
  |  /command
  |  param: query
  |  foo=bar&lorem=ipsum
  |  param: ref
  |  fragment
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

| param: protocol
|   http
| param: host
|   www.google.com
| param: port
|   80
| param: authority
|   user:password@www.google.com:80
| param: userinfo
|   user:password
| param: path
|   /command
| param: query
|   foo=bar&lorem=ipsum
| param: ref
|   fragment
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
  | http
param: protocol
  | http
param: host
  | www.google.com
param: port
  | 80
param: authority
  | user:password@www.google.com:80
param: userinfo
  | user:password
param: path
  | /command
param: query
  | foo=bar&lorem=ipsum
param: ref
  | fragment
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)

| protocol | http |
| host     | www.google.com |
| port     | 80 |
| authority | user:password@www.google.com:80 |
| userinfo | user:password |
| path     | /command |
| query    | foo=bar&lorem=ipsum |
| ref      | fragment |

URL ::= PROTOCOL '://' AUTHORITY
AUTHORITY ::= USERINFO '@' HOST
java.net.URL.set(protocol, host, port, authority, userinfo, path, query, ref)
| http user:password@www.google.com:80/command foo=bar&lorem=ipsum #fragment
| param: protocol
| http
| param: host
| www.google.com
| param: port
| 80
| param: authority
| user:password@www.google.com:80
| param: userinfo
| user:password
| param: path
| /command
| param: query
| foo=bar&lorem=ipsum
| param: ref
| fragment

URL ::= PROTOCOL '://' AUTHORITY PATH '?' QUERY '#' REF
AUTHORITY ::= USERINFO '@' HOST ':' PORT
PROTOCOL ::= 'http'
USERINFO ::= 'user:password'
HOST ::= 'www.google.com'
PORT ::= '80'
PATH ::= '/command'
QUERY ::= 'foo=bar&lorem=ipsum'
REF ::= 'fragment'
Grammar Inference in Python

- We can track variables + values in Python
- We cannot track their dynamic taints
- But we can identify substrings of the input
Grammar Inference in Python

- Start with grammar $START \rightarrow input$
- For each (var, value) we find during execution, where value is a substring of input:
  
  1. In the grammar, replace all occurrences of value by $VAR$
  
  2. Add a new rule $VAR \rightarrow value$
# We store individual variable/value pairs here

```
global the_values
the_values = {}
```

# The current input string

```
global the_input
the_input = None
```

# We record all string variables and values occurring during execution

```
def traceit(frame, event, arg):
    global the_values
    variables = frame.f_locals.keys()

    for var in variables:
        value = frame.f_locals[var]

        # Save all non-trivial string values that also occur in the input
        if type(value) == type('') and len(value) >= 2 and value in the_input:
            the_values[var] = value

    return traceit
```

the_input = "...
sys.settrace(traceit)
program_under_test(the_input)
# Obtain a grammar for a specific input

def get_grammar(input):
    # Here's our initial grammar
    grammar = {"$START": [input]}

    # We obtain a mapping of variables to values
    global the_input
    the_input = input

    global the_values
    the_values = {}

    sys.settrace(traceit)
    o = urlparse(the_input)
    sys.settrace(None)

    # Now for each (VAR, VALUE) found:
    # 1. We search for occurrences of VALUE in the grammar
    # 2. We replace them by $VAR
    # 3. We add a new rule $VAR -> VALUE to the grammar
    while True:
        new_rules = []
        for var in the_values.keys():
            value = the_values[var]
            # Search for occurrences of value in grammar
# Now for each (VAR, VALUE) found:
# 1. We search for occurrences of VALUE in the grammar
# 2. We replace them by $VAR$
# 3. We add a new rule $VAR \rightarrow VALUE$ to the grammar

```python
while True:
    new_rules = []
    for var in the_values.keys():
        value = the_values[var]
        for key in grammar.keys():
            repl_alternatives = grammar[key]
            for j in range(0, len(repl_alternatives)):
                repl = repl_alternatives[j]
                if value in repl:
                    # Found variable value in some grammar nonterminal
                    # Replace value by nonterminal name
                    alt_key = nonterminal(var)
                    repl_alternatives[j] = repl.replace(value, alt_key)
                    new_rules = new_rules + [(var, alt_key, value)]

    if len(new_rules) == 0:
        break  # Nothing to expand anymore

    for (var, alt_key, value) in new_rules:
        # Add new rule to grammar
        grammar[alt_key] = [value]
        # Do not expand this again
        del the_values[var]

return grammar
```
Initial Grammar

'http://www.st.cs.uni-saarland.de/zeller#ref' ->
$START ::= $SCHEME://$NETLOC$URL#$FRAGMENT
$SCHEME ::= http
$NETLOC ::= www.st.cs.uni-saarland.de
$URL ::= $PATH
$PATH ::= /zeller
$FRAGMENT ::= ref
Merging Grammars

- Multiple inputs yield multiple grammars
- Merge these grammars to obtain alternatives
Demo
# Merge two grammars G1 and G2

def merge_grammars(g1, g2):
    merged_grammar = g1
    for key2 in g2.keys():
        repl2 = g2[key2]
        key_found = False
        for key1 in g1.keys():
            repl1 = g1[key1]
            for repl in repl2:
                if key1 == key2:
                    key_found = True
                    if repl not in repl1:
                        # Extend existing rule
                        merged_grammar[key1] = repl1 + [repl]
                
    if not key_found:
        # Add new rule
        merged_grammar[key2] = repl2

    return merged_grammar
Merged Grammars

'http://www.st.cs.uni-saarland.de/zeller#ref' ->
$START ::= $SCHEME://$NETLOC$URL#$FRAGMENT
$SCHEME ::= http
$NETLOC ::= www.st.cs.uni-saarland.de
$URL ::= $PATH
$PATH ::= /zeller
$FRAGMENT ::= ref

∪

'https://www.cispa.saarland:80/bar' ->
$START ::= $SCHEME://$NETLOC$URL
$SCHEME ::= https
$NETLOC ::= www.cispa.saarland:80
$URL ::= $PATH
$PATH ::= /bar
'http://www.st.cs.uni-saarland.de/zeller#ref' ->
$START ::= $SCHEME://$NETLOC$URL#$FRAGMENT
$SCHEME ::= http
$NETLOC ::= www.st.cs.uni-saarland.de
$URL ::= $PATH
$PATH ::= /zeller
$FRAGMENT ::= ref

'https://www.cispa.saarland:80/bar' ->
$START ::= $SCHEME://$NETLOC$URL
$SCHEME ::= https
$NETLOC ::= www.cispa.saarland:80
$URL ::= $PATH
$PATH ::= /bar

'http://foo@google.com:8080/bar?q=r#ref2' ->
$URL ::= $PATH
$START ::= $SCHEME://$NETLOC$URL?$QUERY#$FRAGMENT
$PATH ::= /bar
$QUERY ::= q=r
$NETLOC ::= foo@google.com:8080
$FRAGMENT ::= ref2
$SCHEME ::= http
Merged Grammars

Merged grammar ->
$URL ::= $PATH
$START ::= $SCHEME://$NETLOC$URL#$FRAGMENT | $SCHEME://$NETLOC$URL | $SCHEME://$NETLOC$URL?$QUERY#$FRAGMENT
$PATH ::= /zeller | /bar
$QUERY ::= q=r
$NETLOC ::= www.st.cs.uni-saarland.de | www.cispa.saarland:80 | foo@google.com:8080
$FRAGMENT ::= ref | ref2
$SCHEME ::= http | https
Fuzzing

Fuzzing ->
https://www.cispa.saarland:80/zeller
https://www.cispa.saarland:80/bar#ref
http://www.st.cs.uni-saarland.de/zeller#ref2
http://www.cispa.saarland:80/bar#ref
https://www.st.cs.uni-saarland.de/zeller#ref
http://foo@google.com:8080/bar
http://www.cispa.saarland:80/bar#ref
https://www.st.cs.uni-saarland.de/bar#ref2
http://www.st.cs.uni-saarland.de/zeller#ref
...

INI Files

[Application]
Version = 0.5
WorkingDir = /tmp/mydir/
[User]
User = Bob
Password = 12345
JSON Input

```json
{
  "v": true,
  "x": 25,
  "y": -36,
  ...
}
```
AUTOGRAM
Grammars

• give insights into the *structure of inputs*
  → reverse engineering
  → writing tests
  → writing parsers

• first technique to mine input grammars from programs

fully automatic • scalable • practical
List of file formats

From Wikipedia, the free encyclopedia

This is a dynamic list and may never be able to satisfy particular standards for completeness. You can help by expanding it with reliably sourced entries.

See also: List of filename extensions

This is a list of file formats used by computers, organized by type. Filename extensions are usually noted in parentheses if they differ from the format name or abbreviation. Many operating systems do not limit filenames to a single extension shorter than 4 characters, as was common with some operating systems that supported the FAT file system. Examples of operating systems that do not impose this limit include Unix-like systems. Also, Microsoft Windows NT, 95, 98, and Me do not have a three character limit on extensions for 32-bit or 64-bit applications on file systems other than pre-Windows 95/Windows NT 3.5 versions of the FAT file system. Some filenames are given extensions longer than three characters.

Some file formats may be listed twice or more. An example is the .b file.
Today's Contents

- **Fuzzing 101**
  - Simple fuzzing techniques generating *random inputs* to programs

- **Grammar-Based Fuzzing**
  - Structured fuzzing techniques using *grammars* and models

- **Inferring Grammars**
  - Inferring input grammars so you can fuzz arbitrary programs
Today's Contents

- **Fuzzing 101**: Simple fuzzing techniques generating *random inputs* to programs
- **Grammar-Based Fuzzing**: Structured fuzzing techniques using *grammars* and models
- **Inferring Grammars**: Inferring input grammars so you can fuzz arbitrary programs
Current Research

• Dynamic taints from C and Java programs

• Active + sample-free learning of grammars

• Guiding fuzzing towards code coverage

• Integration with symbolic testing

• Build the world’s best fuzzing platform!

M. Höschele
N. Havrikov
A. Kampmann
Research Opportunities

- What is the input language of a program?
- How can I leverage input structure to cover, understand, prevent, inputs, code, behaviors?
- Hundreds of open issues! (and theses)
Grammar-Based Fuzzing

```javascript
var haystack = "foo";
var re_text = "~foo";
haystack += "x";
re_text += "(x)";
var re = new RegExp(re_text);
re. test(haystack);
RegExp. input = Number();
print(RegExp.$1);
```

Current Research

- Dynamic taints from C and Java programs
- Active + sample-free learning of grammars
- Guiding fuzzing towards code coverage
- Integration with symbolic testing
- Build the world’s best fuzzing platform!

Research Opportunities

- What is the input language of a program?
- How can I leverage input structure to
  - cover
  - understand
  - prevent
- Hundreds of open issues! (and theses)