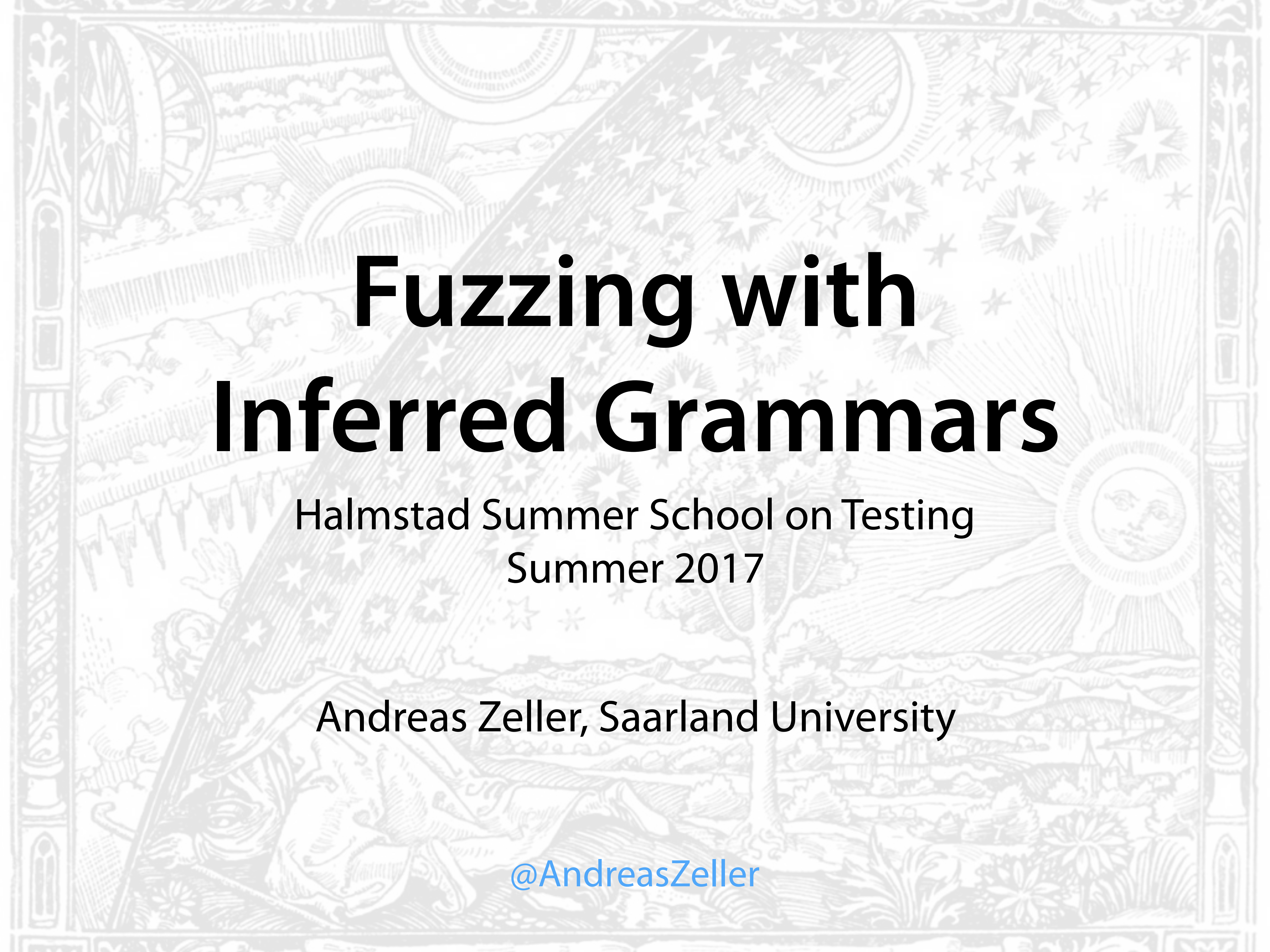


Fuzzing with Inferred Grammars

Halmstad Summer School on Testing
Summer 2017

Andreas Zeller, Saarland University

@AndreasZeller

Saarbrücken

CISPA Saarbrücken

© av-a Veauthier Architekten

~~Fuzzing with Inferred Grammars~~

Halmstad Summer School on Testing
Summer 2017

Andreas Zeller, Saarland University

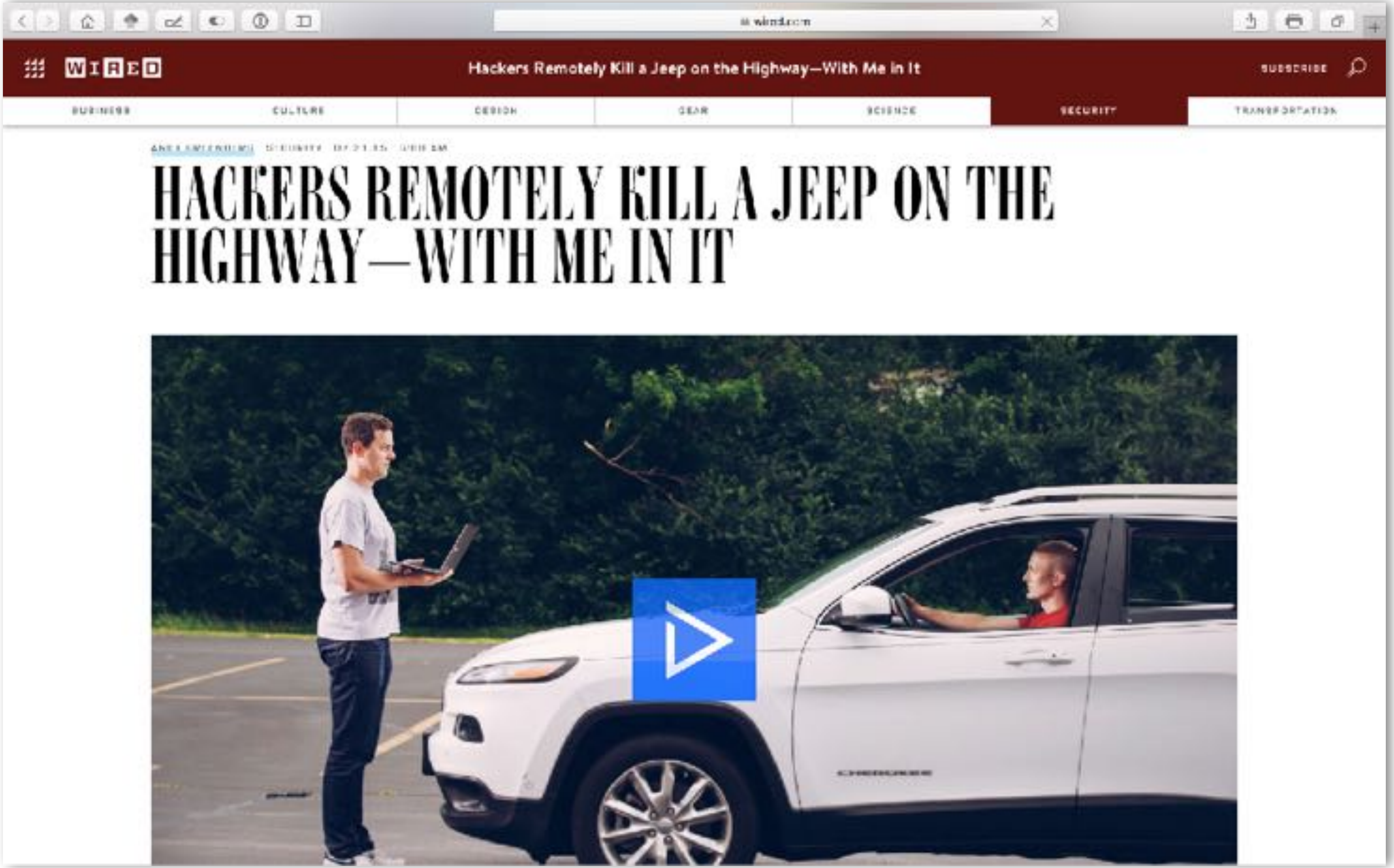
@AndreasZeller

~~Fuzzing with Inferred Grammars~~

Making $50,000/Month Fuzzing Software

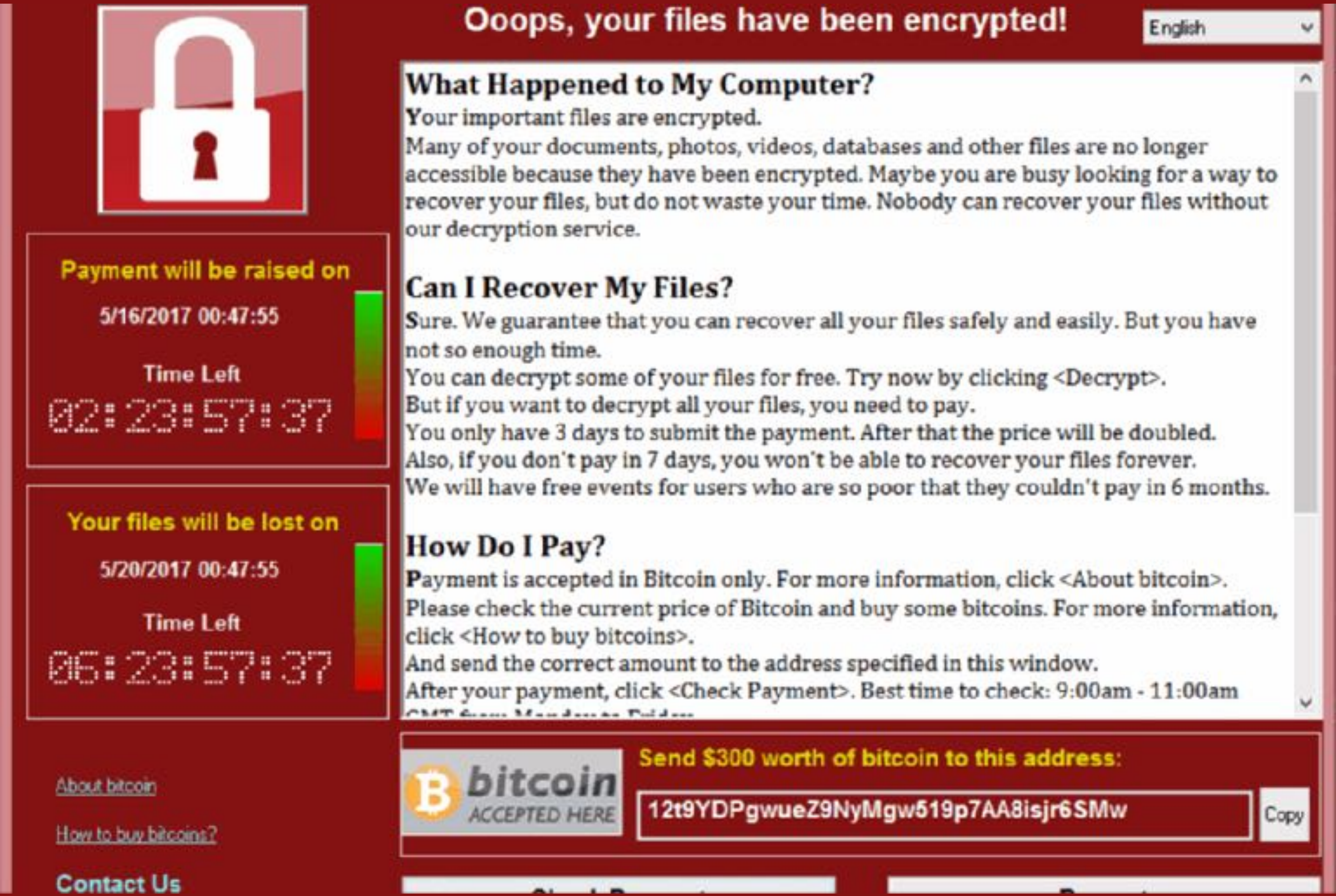
Halmstad Summer School on Testing
Summer 2017

Andreas Zeller, Saarland University
@AndreasZeller



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

Payment will be raised on

5/16/2017 00:47:55

Time Left

02:23:57:37

Your files will be lost on

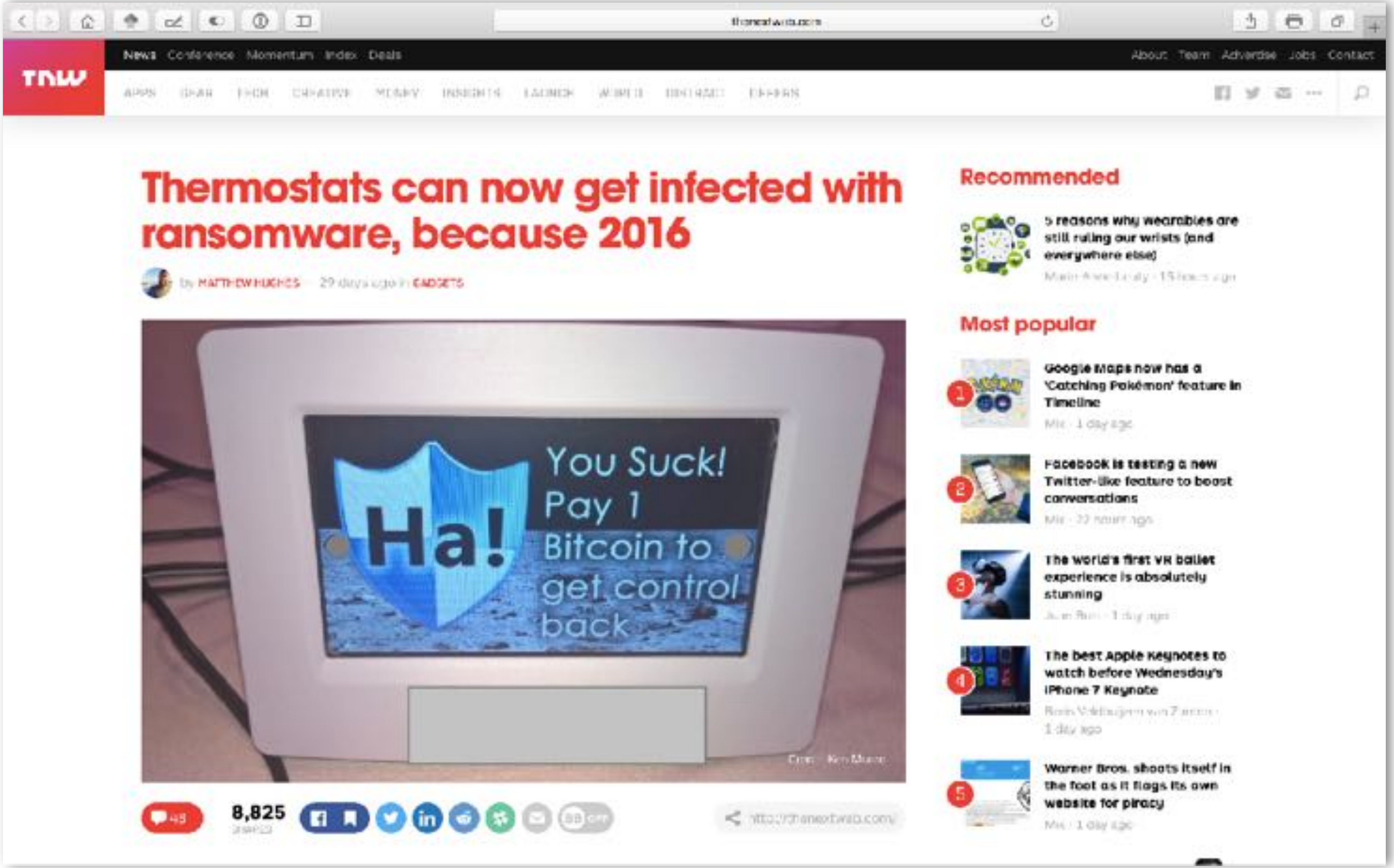
5/20/2017 00:47:55

Time Left

06:23:57:37

Send $300 worth of bitcoin to this address:

12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw



thenextweb.com
TNW
News Conference Momentum Index Deals
About Team Advertise Jobs Contact
APPS GEAR TECH CREATIVE MONEY INSIGHTS LAUNCH WORLD DISTRACT OFFERS

Thermostats can now get infected with ransomware, because 2016

by MATTHEW HUGHES — 29 days ago in GADGETS

Ha!
You Suck!
Pay 1
Bitcoin to
get control
back

Credit: Ken Munro

48
8,825
SHARES
http://thenextweb.com/

Recommended

5 reasons why wearables are still ruling our wrists (and everywhere else)
[illegible] · 15 hours ago

Most popular

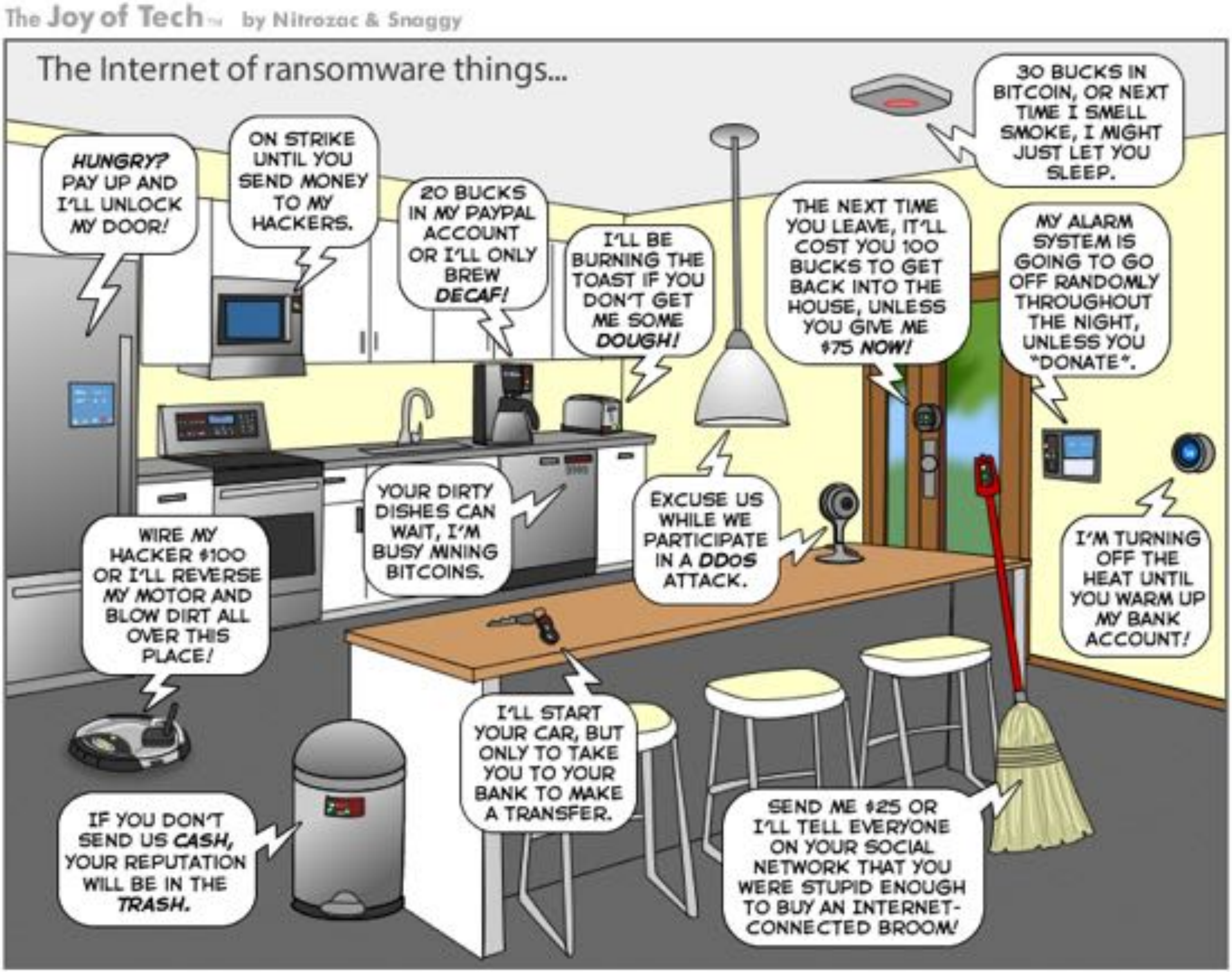
1. Google Maps now has a 'Catching Pokémon' feature in Timeline
Mix · 1 day ago

2. Facebook is testing a new Twitter-like feature to boost conversations
Mix · 22 hours ago

3. The world's first VR ballet experience is absolutely stunning
Juan Buis · 1 day ago

4. The best Apple Keynotes to watch before Wednesday's iPhone 7 Keynote
Boris Veldhuijzen van Zanten · 1 day ago

5. Warner Bros. shoots itself in the foot as it flags its own website for piracy
Mix · 1 day ago

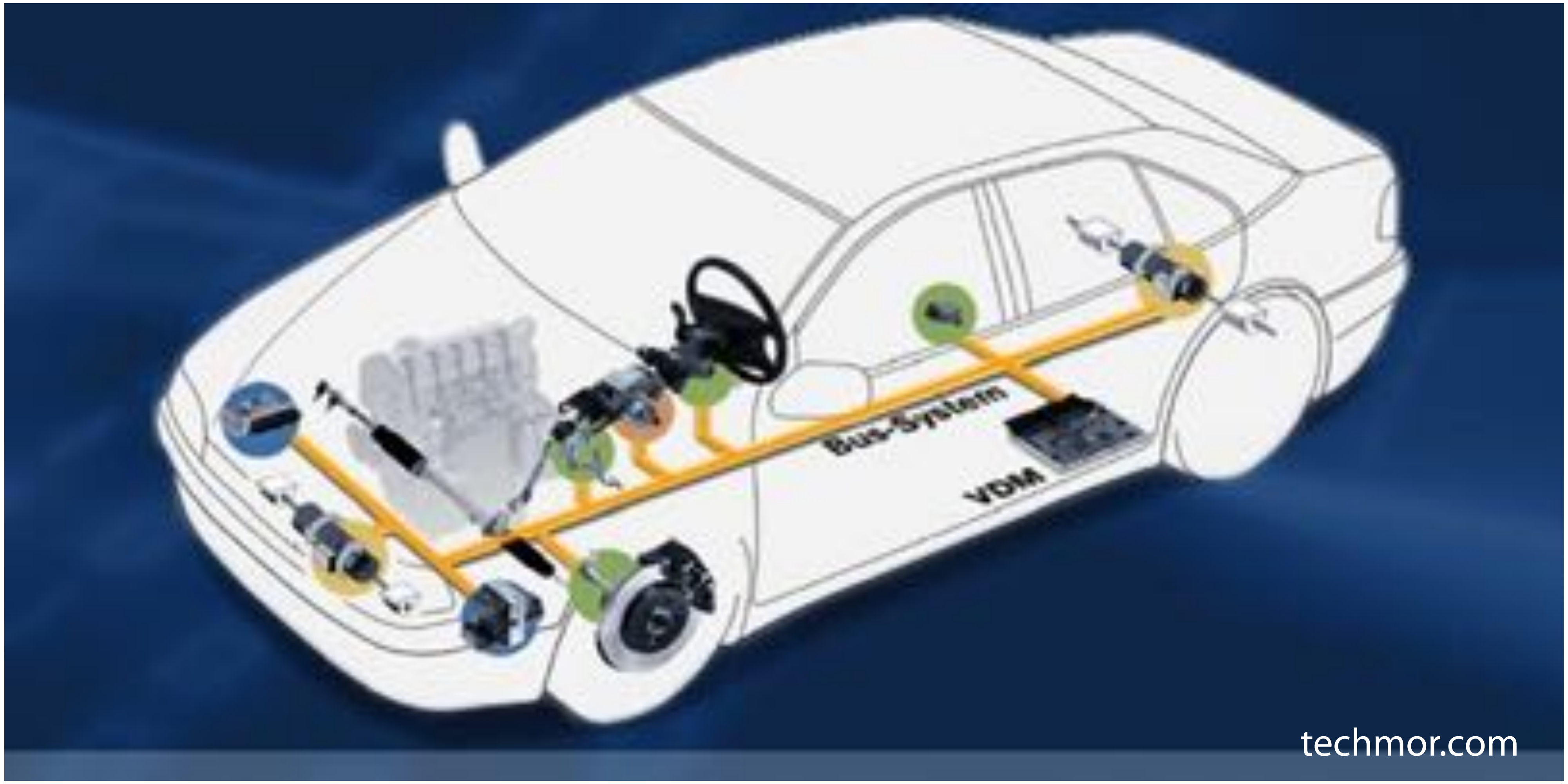

The Joy of Tech™ by Nitrozac & Snaggy
The Internet of ransomware things...
HUNGRY? PAY UP AND I'LL UNLOCK MY DOOR!
ON STRIKE UNTIL YOU SEND MONEY TO MY HACKERS.
20 BUCKS IN MY PAYPAL ACCOUNT OR I'LL ONLY BREW DECAF!
I'LL BE BURNING THE TOAST IF YOU DON'T GET ME SOME DOUGH!
THE NEXT TIME YOU LEAVE, IT'LL COST YOU 100 BUCKS TO GET BACK INTO THE HOUSE, UNLESS YOU GIVE ME $75 NOW!
30 BUCKS IN BITCOIN, OR NEXT TIME I SMELL SMOKE, I MIGHT JUST LET YOU SLEEP.
MY ALARM SYSTEM IS GOING TO GO OFF RANDOMLY THROUGHOUT THE NIGHT, UNLESS YOU "DONATE".
YOUR DIRTY DISHES CAN WAIT, I'M BUSY MINING BITCOINS.
EXCUSE US WHILE WE PARTICIPATE IN A DDoS ATTACK.
I'M TURNING OFF THE HEAT UNTIL YOU WARM UP MY BANK ACCOUNT!
WIRE MY HACKER $100 OR I'LL REVERSE MY MOTOR AND BLOW DIRT ALL OVER THIS PLACE!
I'LL START YOUR CAR, BUT ONLY TO TAKE YOU TO YOUR BANK TO MAKE A TRANSFER.
IF YOU DON'T SEND US CASH, YOUR REPUTATION WILL BE IN THE TRASH.
SEND ME $25 OR I'LL TELL EVERYONE ON YOUR SOCIAL NETWORK THAT YOU WERE STUPID ENOUGH TO BUY AN INTERNET-CONNECTED BROOM!

External Attacks

Program

- Some external event
  causes a *change in program behavior*

Highjacking a Car



techmor.com

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
- Any simple test would find that!

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*

But *no guarantee* of absence of bugs

This Course

- Introduces you to **automated techniques for security testing**
- Enables you to **implement** and **use** such techniques
- Aim: **Smart ways to break systems**

Programming Language

By Doc Searls – 2006oscon_203.JPG, CC BY-SA 2.0
https://commons.wikimedia.org/w/index.php?curid=4974869

Course 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

Course 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

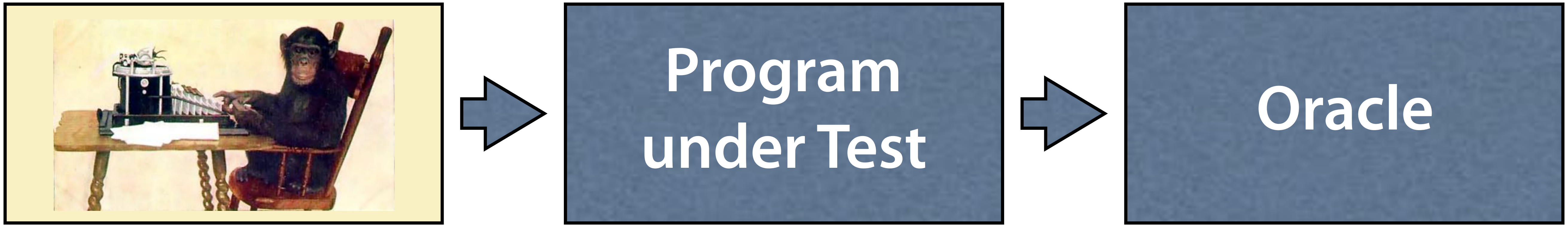
so you can fuzz arbitrary programs

Monkey Testing

Youtube

Infinite Monkey Theorem

Random Testing

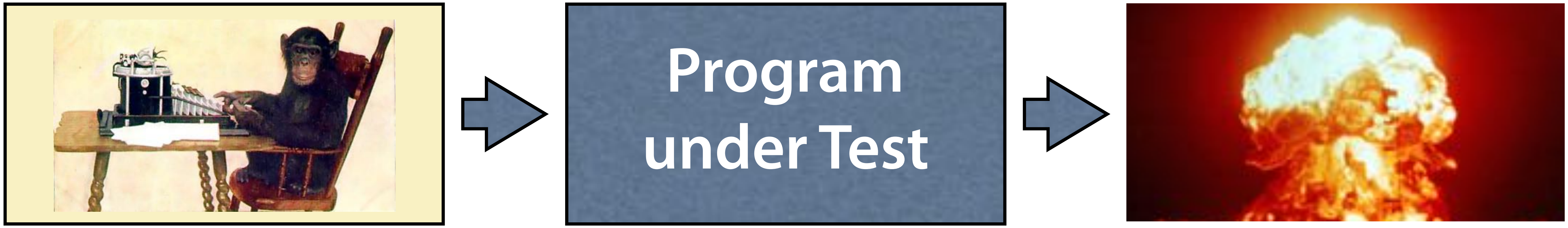


Program under Test

Oracle

Fuzzing

Random Testing at the System Level



Program under Test

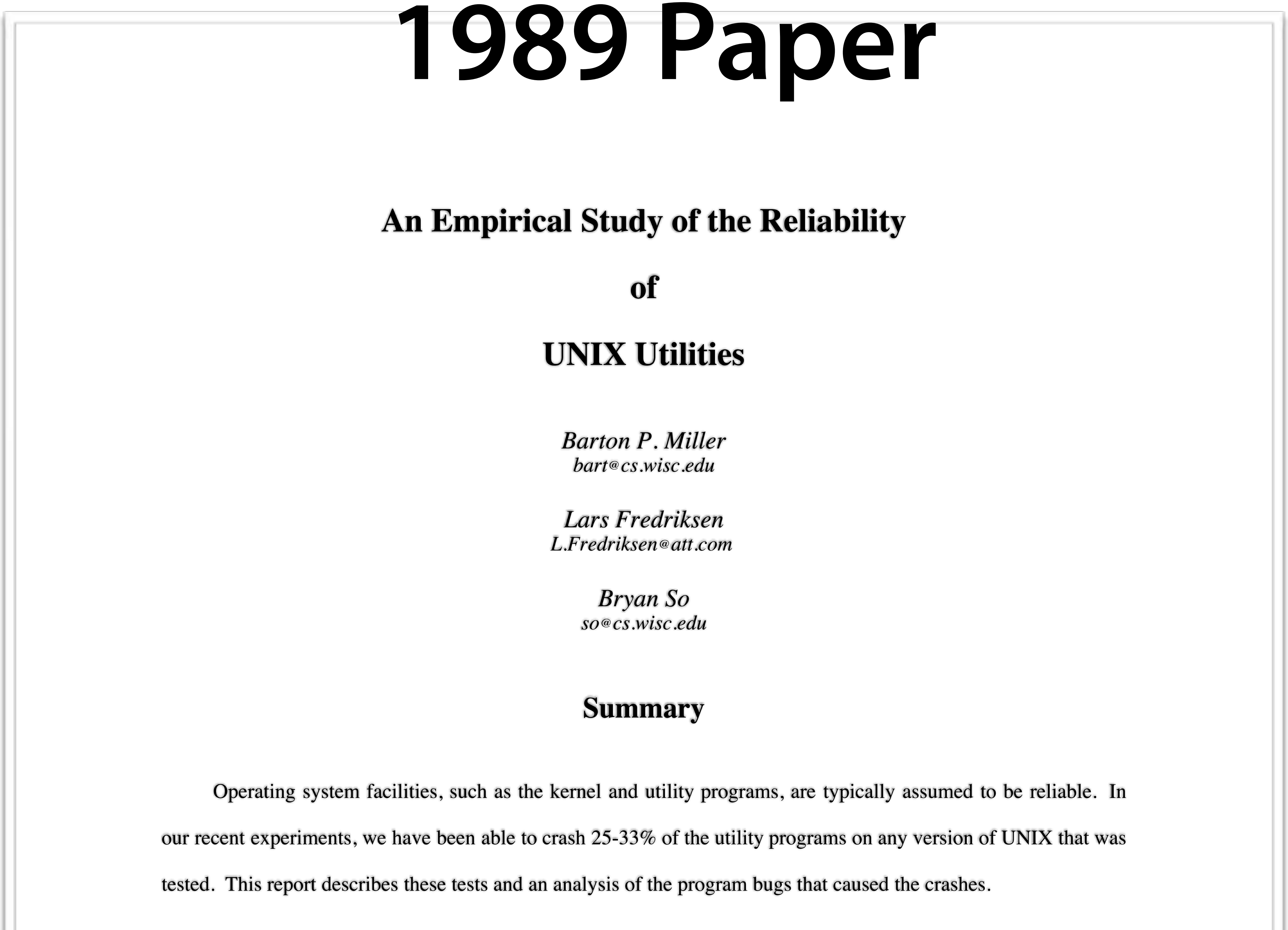
“ab’d&gfdfggg”

Fuzzing

Random Testing at the System Level

Barton P. Miller

1989 Paper



An Empirical Study of the Reliability of UNIX Utilities

Barton P. Miller
bart@cs.wisc.edu

Lars Fredriksen
L.Fredriksen@att.com

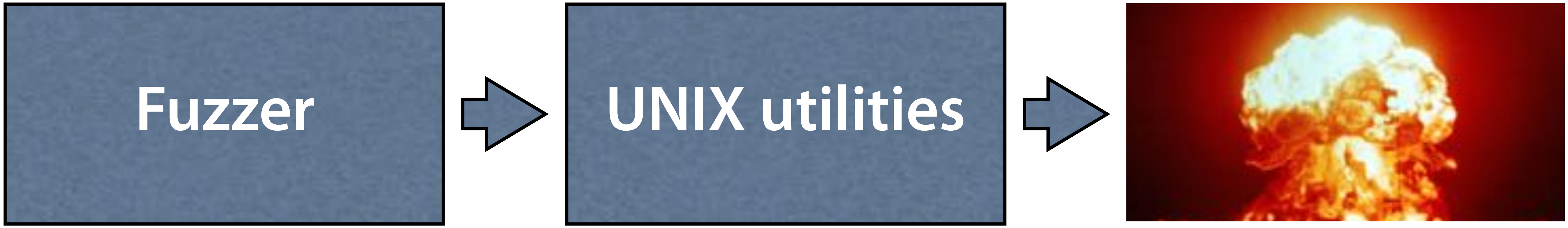
Bryan So
so@cs.wisc.edu

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&gfdfggg”

grep • sh • sed …

25%–33%

Fuzzer Output

```
[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!
AxB"YXRS@!Kd6;wtAMefFWM(`|J_<1~o}z3K(CCzRH
JIIvHz>_*.\>JrlU32~eGP?lR=bF3+;y$3lodQ<B89!
5"W2fK*vE7v{')KC-i,c{<[~m!]o;{.'}Gj\(X}
EtYetrpbY@aGZ1{P!AZU7x#4(Rtn!q4nCwqol^y6}0|
Ko=*JK~;zMKV=9Nai:wxu{J&UV#HaU)*BiC<),`+t*gka<W=Z.
%T5WGHZpI30D<Pq>&]BS6R&j?#tP7iaV}-}`\?[_[Z^LBMPG-
FKj'\xwuZ1=Q`^`5,$N$Q@[!CuRzJ2D|vBy!^zkhdf3C5PAkR?
V hn|
3='i2Qx]D$qs4O`1@fevnG'2\11Vf3piU37@55ap\zIyl"'f,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}
r[Scun&sBCS,T[/vY'pduwgzDlVNy7'rnzxNwI)(ynBa>%|
b`;`9fG]P_0hdG~$@6 3]KAeEnQ7lU)3Pn,0)G/6N-wyzj/
MTd#A;r
```

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
```

fuzzer.py

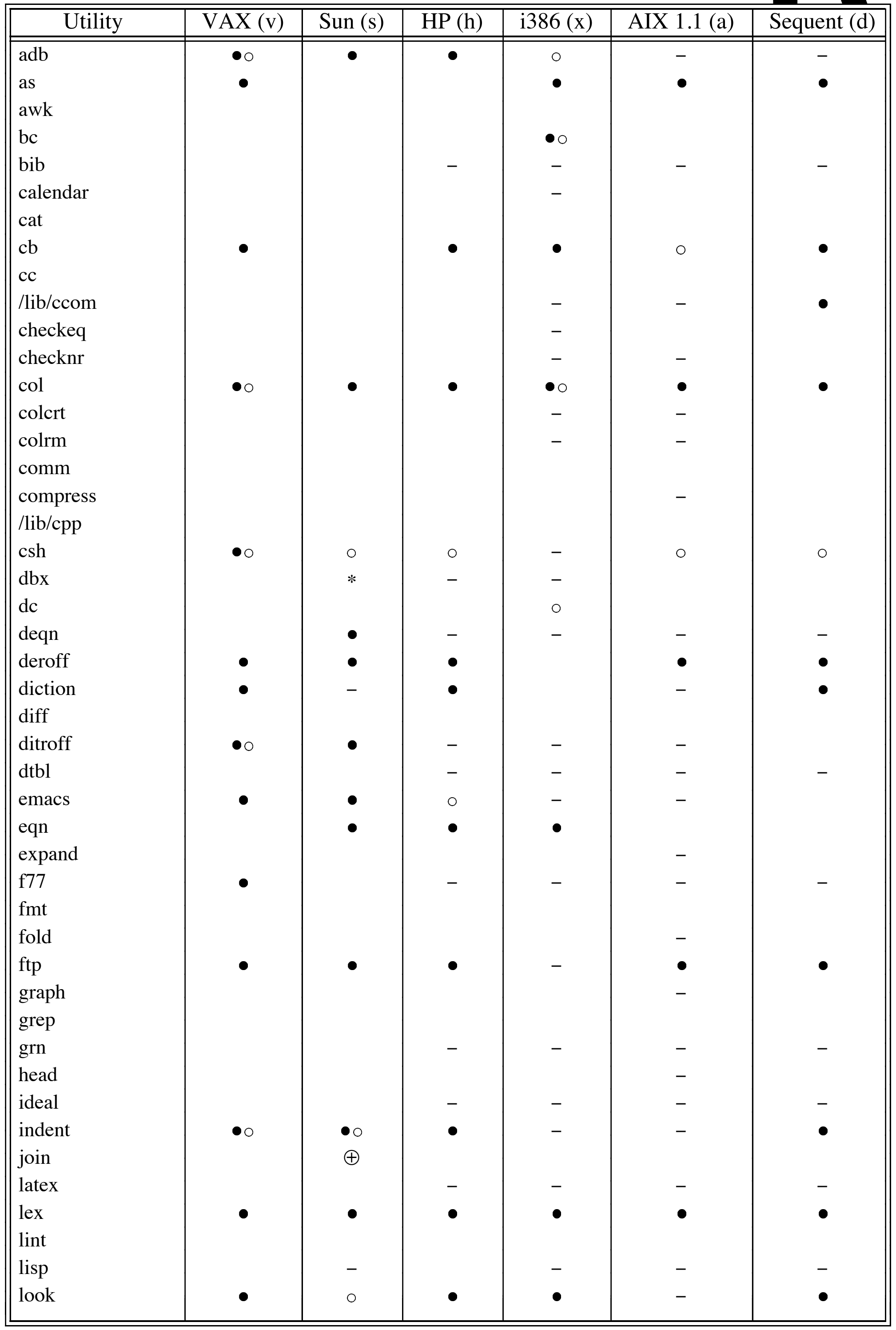
```
import random

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

Utility	VAX (v)	Sun (s)	HP (h)	i386 (x)	AIX 1.1 (a)	Sequent (d)
adb	●○	●	●	○	–	–
as	●			●	●	●
awk						
bc				●○		
bib			–	–	–	–
calendar				–		
cat						
cb	●		●	●	○	●
cc						
/lib/ccom				–	–	●
checkeq				–		
checknr				–	–	
col	●○	●	●	●○	●	●
colcrt				–	–	
colrm				–	–	
comm						
compress					–	
/lib/cpp						
csh	●					

deroff	●	●	●		●	●
diction	●	–	●		–	●
diff						
ditroff	●○	●	–	–	–	
dtbl			–	–	–	–
emacs	●	●	○	–	–	
eqn		●	●	●		
expand					–	
f77	●		–	–	–	–
fmt						
fold					–	
ftp	●	●	●	–	●	●
graph					–	
grep						
grn			–	–	–	–
head					–	
ideal			–	–	–	–
indent	●○	●○	●	–	–	●
join		⊕				
latex			–	–	–	–
lex	●	●	●	●	●	●
lint						
lisp		–		–	–	–
look	●	○	●	●	–	●

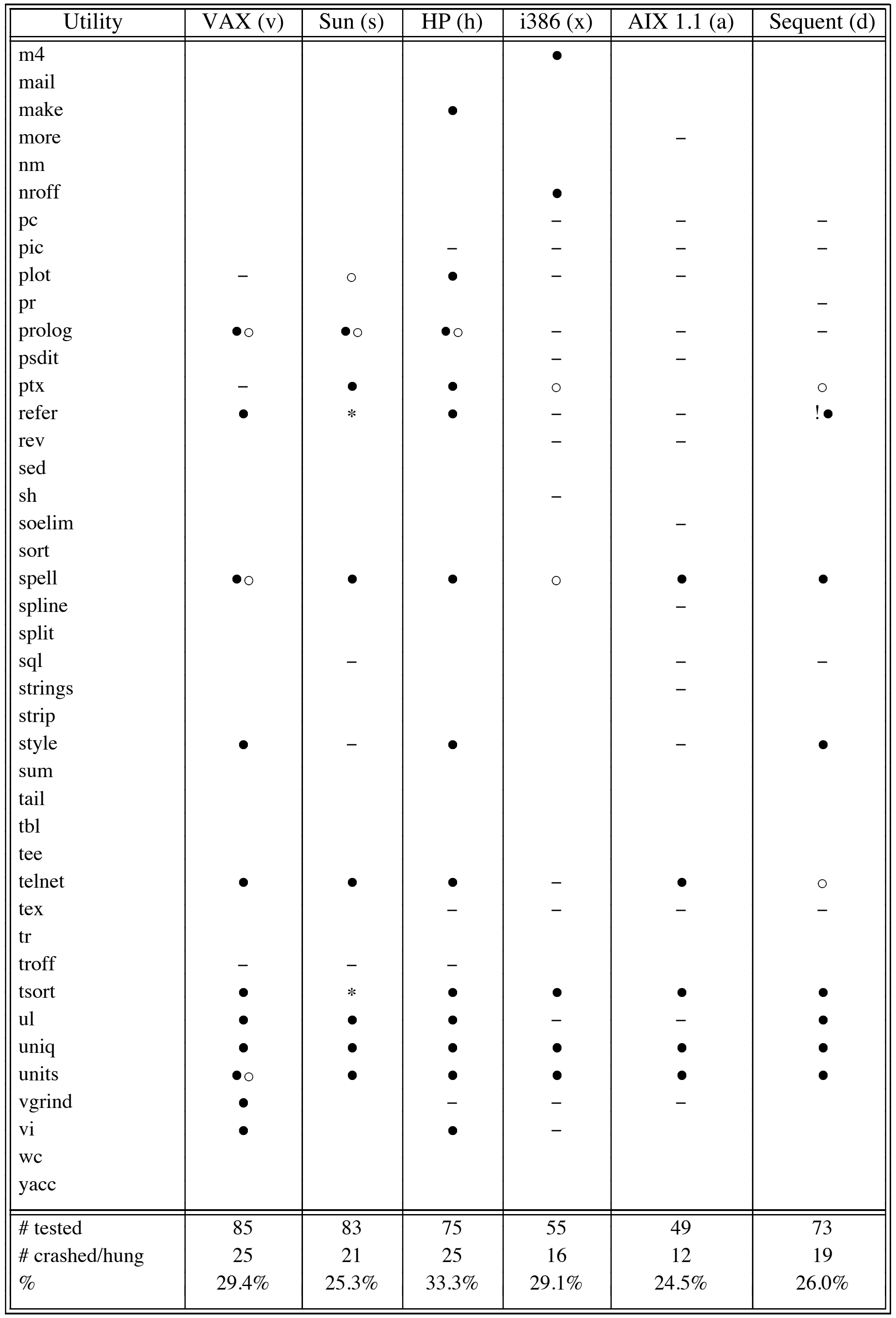
Results



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

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.



Utility	VAX (v)	Sun (s)	HP (h)	i386 (x)	AIX 1.1 (a)	Sequent (d)
m4				●		
mail						
make			●			
more					–	
nm						
nroff				●		
pc				–	–	–
pic			–	–	–	–
plot	–	○	●	–	–	
pr						–
prolog	●○	●○	●○	–	–	–
psdit				–	–	
ptx	–	●	●	○		○
refer	●	*	●	–	–	!●
rev				–	–	
sed						
sh				–		
soelim					–	
sort						
spell	●○	●	●	○	●	●
spline					–	
split						
sql		–			–	–
strings					–	
strip						
style	●	–	●		–	●
sum						
tail						
tbl						
tee						
telnet	●	●	●	–	●	○
tex			–	–	–	–
tr						
troff	–	–	–			
tsort	●	*	●	●	●	●
ul	●	●	●	–	–	●
uniq	●	●	●	●	●	●
units	●○	●	●	●	●	●
vgrind	●		–	–	–	
vi	●		●	–		
wc						
yacc						
# 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…

Pointers and Arrays

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

Not checking Return Codes

```
char rdc()
{
    char lastc;

    do {
        lastc = getchar();
    } while (lastc != ’ ’ ||
             lastc != ’\t’);

    return (lastc);
}
```

And more…

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

```
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

Course 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

Course Contents

Fuzzing 101

Simple **fuzzing** techniques

generating *random inputs* to programs

Grammar-Based Fuzzing

Structured fuzzing techniques

using *grammars* and models

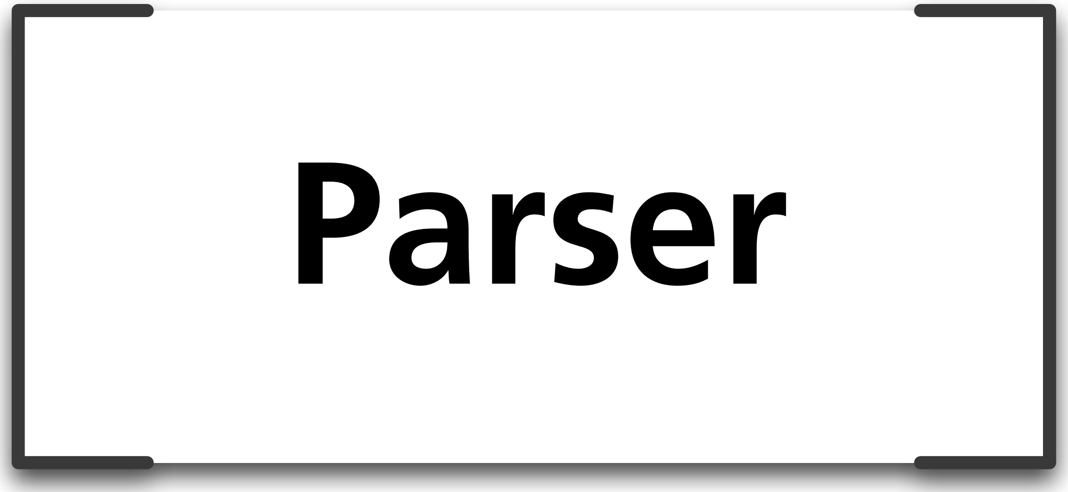
Inferring Grammars

Inferring input grammars

so you can fuzz arbitrary programs

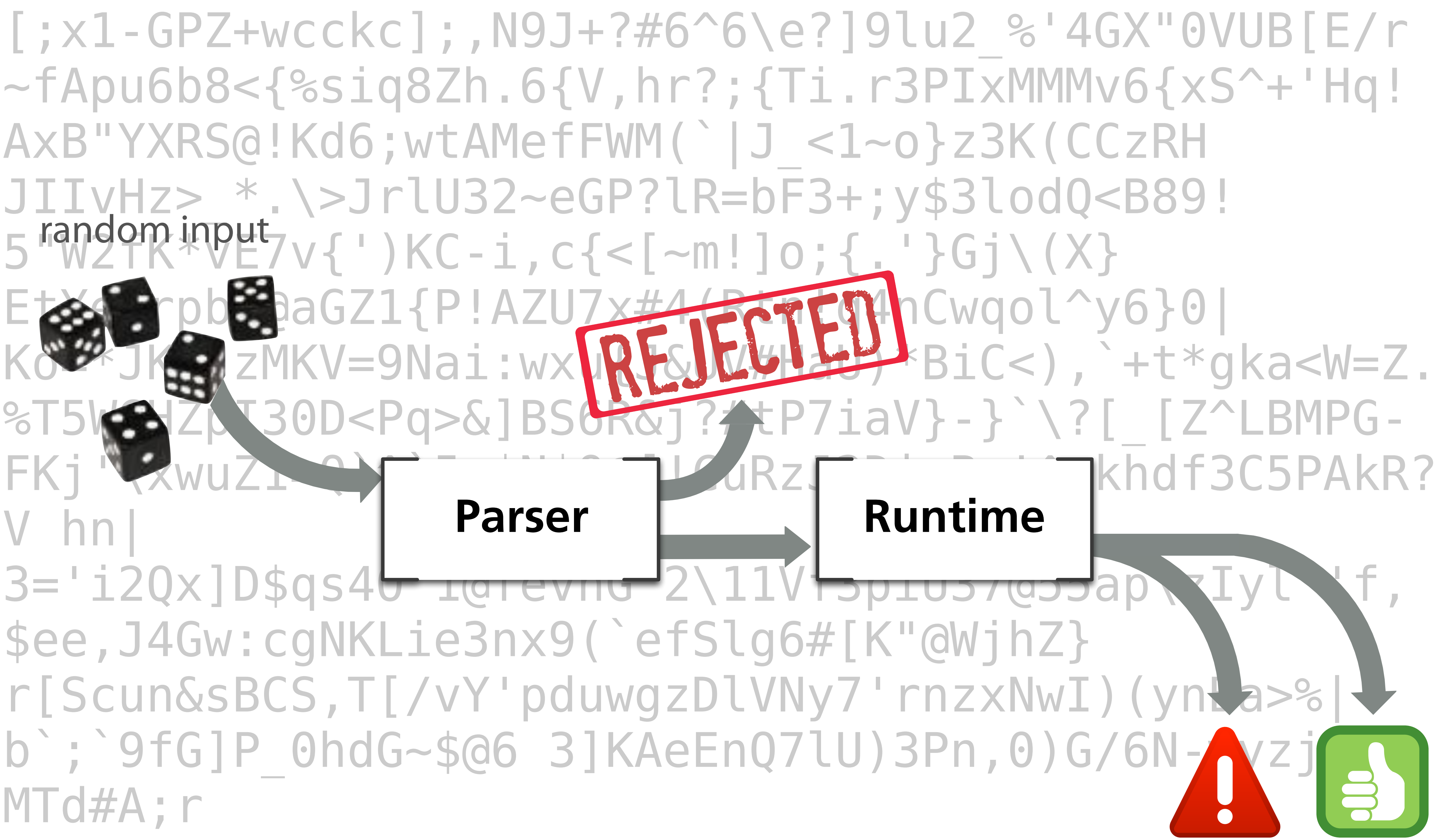
Grammar Fuzzing

- Suppose you want to test a *parser* – to compile and execute a program

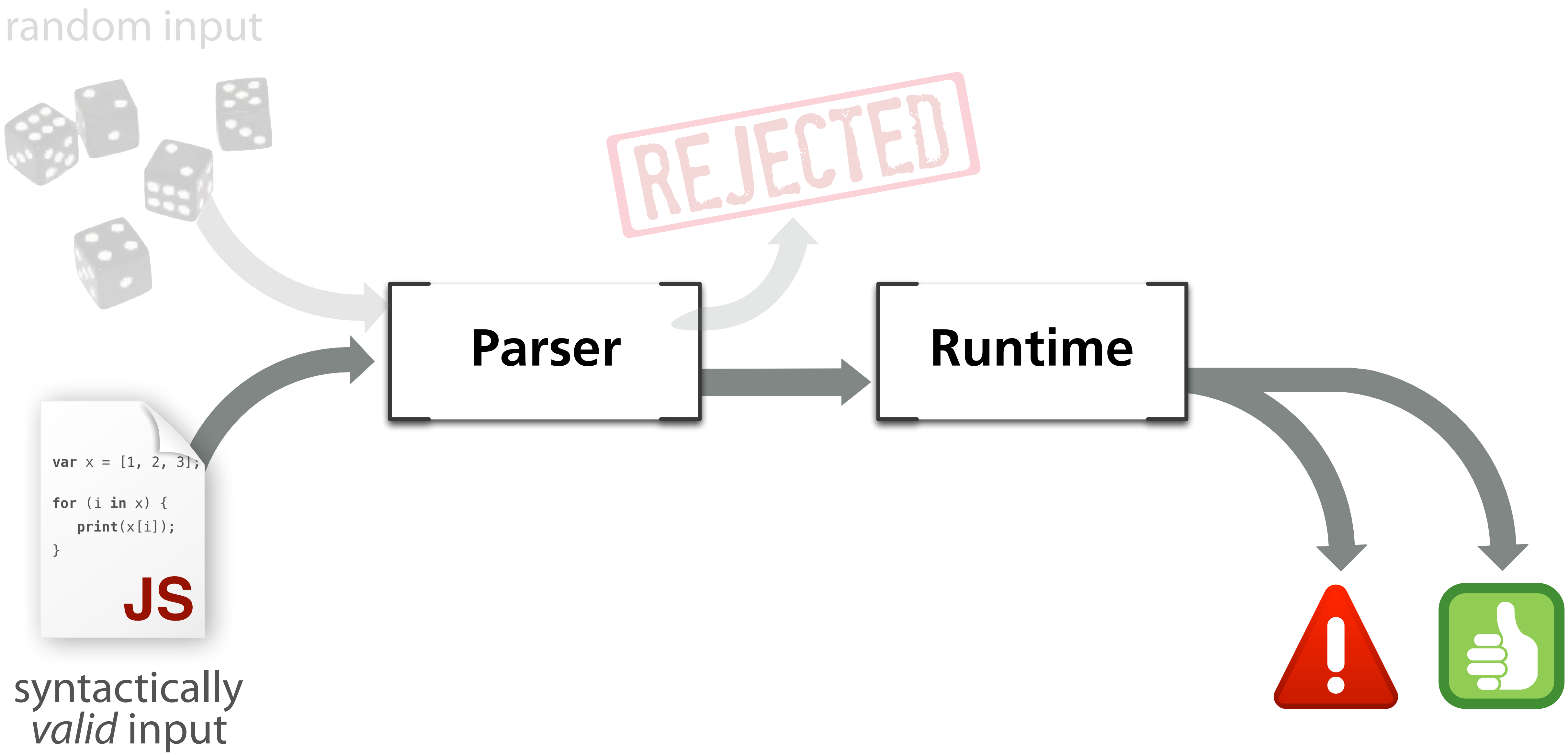


Parser

Grammar Fuzzing


[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!
AxB"YXRS@!Kd6;wtAMefFWM(`|J_<1~o}z3K(CCzRH
JIIvHz> *.\>JrlU32~eGP?lR=bF3+;y$3lodQ<B89!
random input
REJECTED
Parser
Runtime
V hn|
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}
MTd#A;r

Grammar Fuzzing



random input

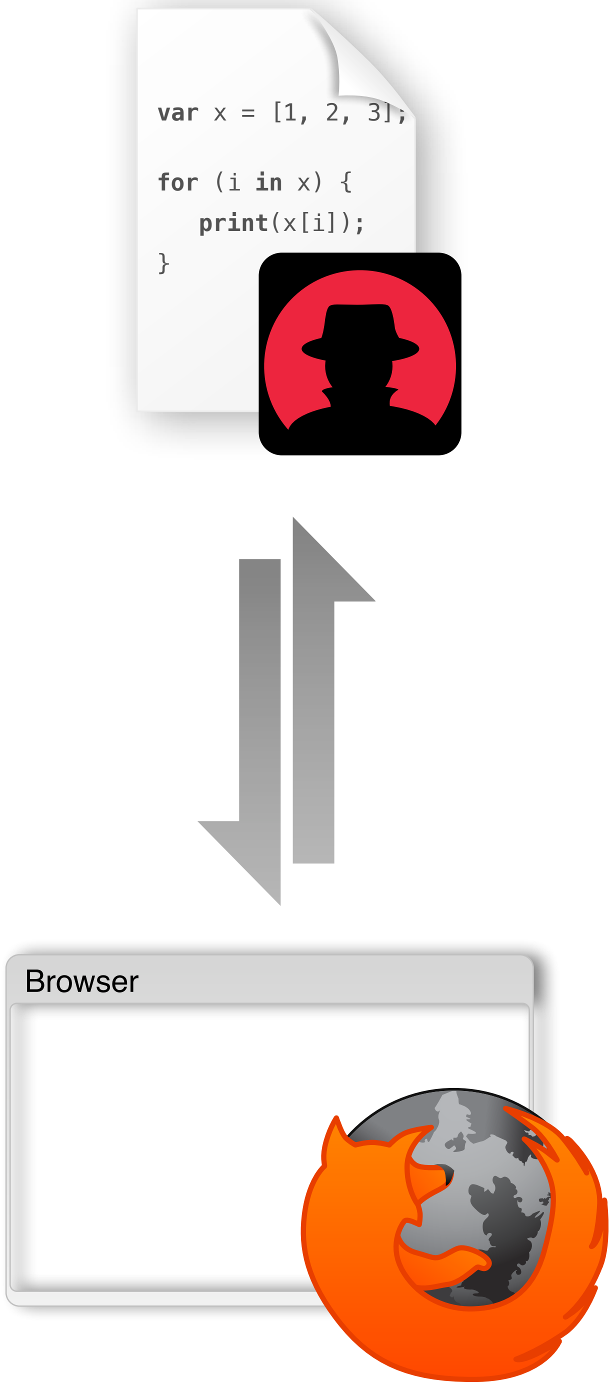
```
var x = [1, 2, 3];

for (i in x) {
   print(x[i]);
}
```

syntactically *valid* input

LangFuzz

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

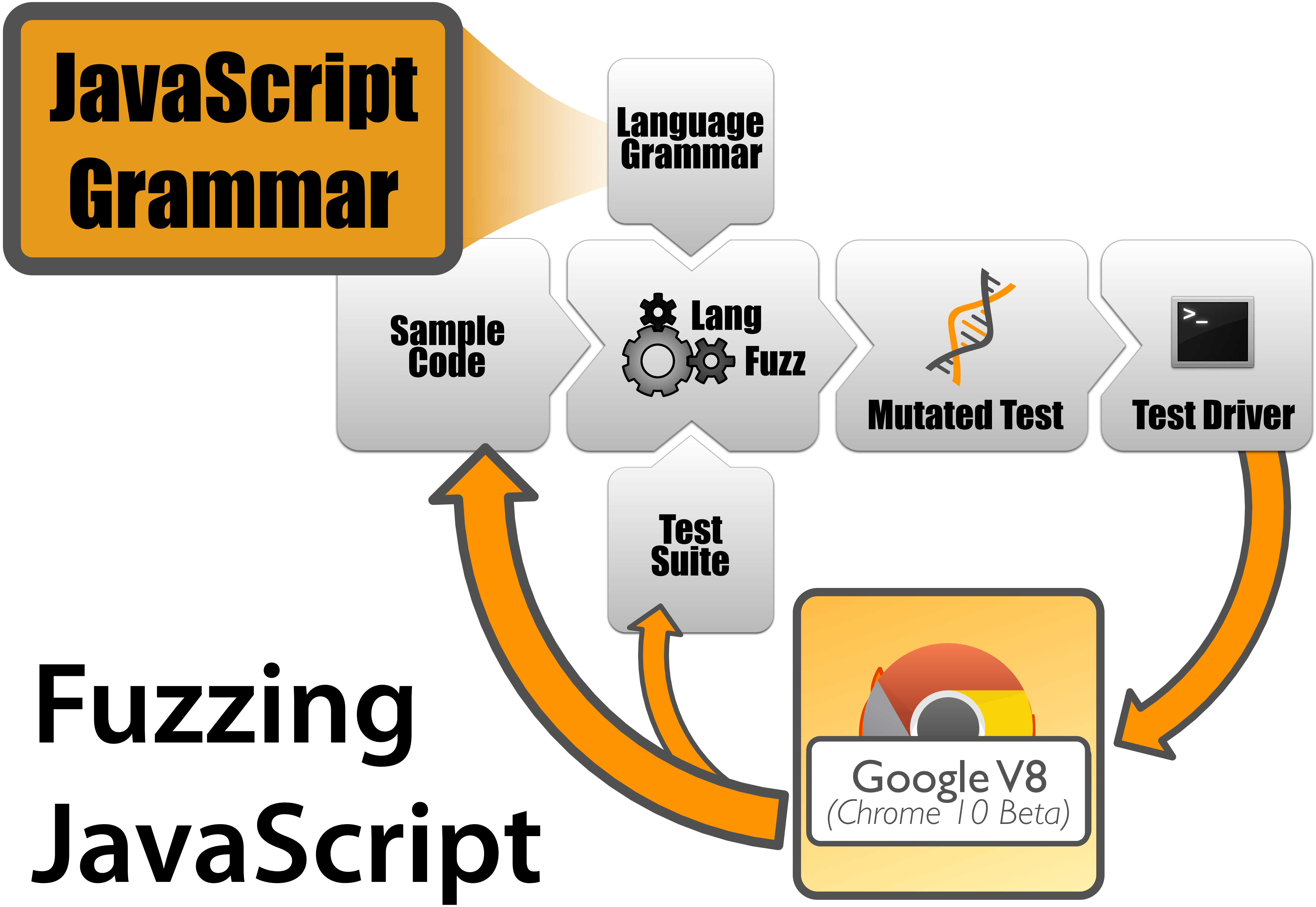
JavaScript as Domain



```
var x = [1, 2, 3];

for (i in x) {
    print(x[i]);
}
```

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

Fuzzing JavaScript



JavaScript Grammar

If Statement

IfStatement^{full} ⇒
  `if` *ParenthesizedExpression* *Statement*^{full}
\| `if` *ParenthesizedExpression* *Statement*^{noShortIf} `else` *Statement*^{full}

IfStatement^{noShortIf} ⇒ `if` *ParenthesizedExpression* *Statement*^{noShortIf} `else` *Statement*^{noShortIf}

Switch Statement

SwitchStatement ⇒
  `switch` *ParenthesizedExpression* `{ }`
\| `switch` *ParenthesizedExpression* `{` *CaseGroups* *LastCaseGroup* `}`

CaseGroups ⇒
  «empty»
\| *CaseGroups* *CaseGroup*

CaseGroup ⇒ *CaseGuards* *BlockStatementsPrefix*

LastCaseGroup ⇒ *CaseGuards* *BlockStatements*

CaseGuards ⇒
  CaseGuard
\| *CaseGuards* *CaseGuard*

CaseGuard ⇒

Fuzzing with Grammars

- Want to encode a *grammar* to produce arithmetic expressions as *strings*
- $START expands into $EXPR, which can expand into $TERM, $EXPR + $TERM, etc.

```
$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

```
$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

$EXPR

```
$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

```
$EXPR + $TERM
```

```
$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

```
$EXPR + $FACTOR
```

```
$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

```
$TERM + $FACTOR
```

```
$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

$FACTOR + $FACTOR

```
$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

```
$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

$INTEGER + $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

```
$DIGIT + $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

```
2 + $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

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

IfStatement^{full} ⇒

  `if` *ParenthesizedExpression* *Statement*^{full}

| **`if`** *ParenthesizedExpression* *Statement*^{noShortIf} **`else`** *Statement*^{full}

IfStatement^{noShortIf} ⇒ **`if`** *ParenthesizedExpression* *Statement*^{noShortIf} **`else`** *Statement*^{noShortIf}

Switch Statement

SwitchStatement ⇒

  `switch` *ParenthesizedExpression* **`{ }`**

| **`switch`** *ParenthesizedExpression* **`{`** *CaseGroups* *LastCaseGroup* **`}`**

CaseGroups ⇒

  «empty»

| *CaseGroups* *CaseGroup*

CaseGroup ⇒ *CaseGuards* *BlockStatementsPrefix*

LastCaseGroup ⇒ *CaseGuards* *BlockStatements*

CaseGuards ⇒

  CaseGuard

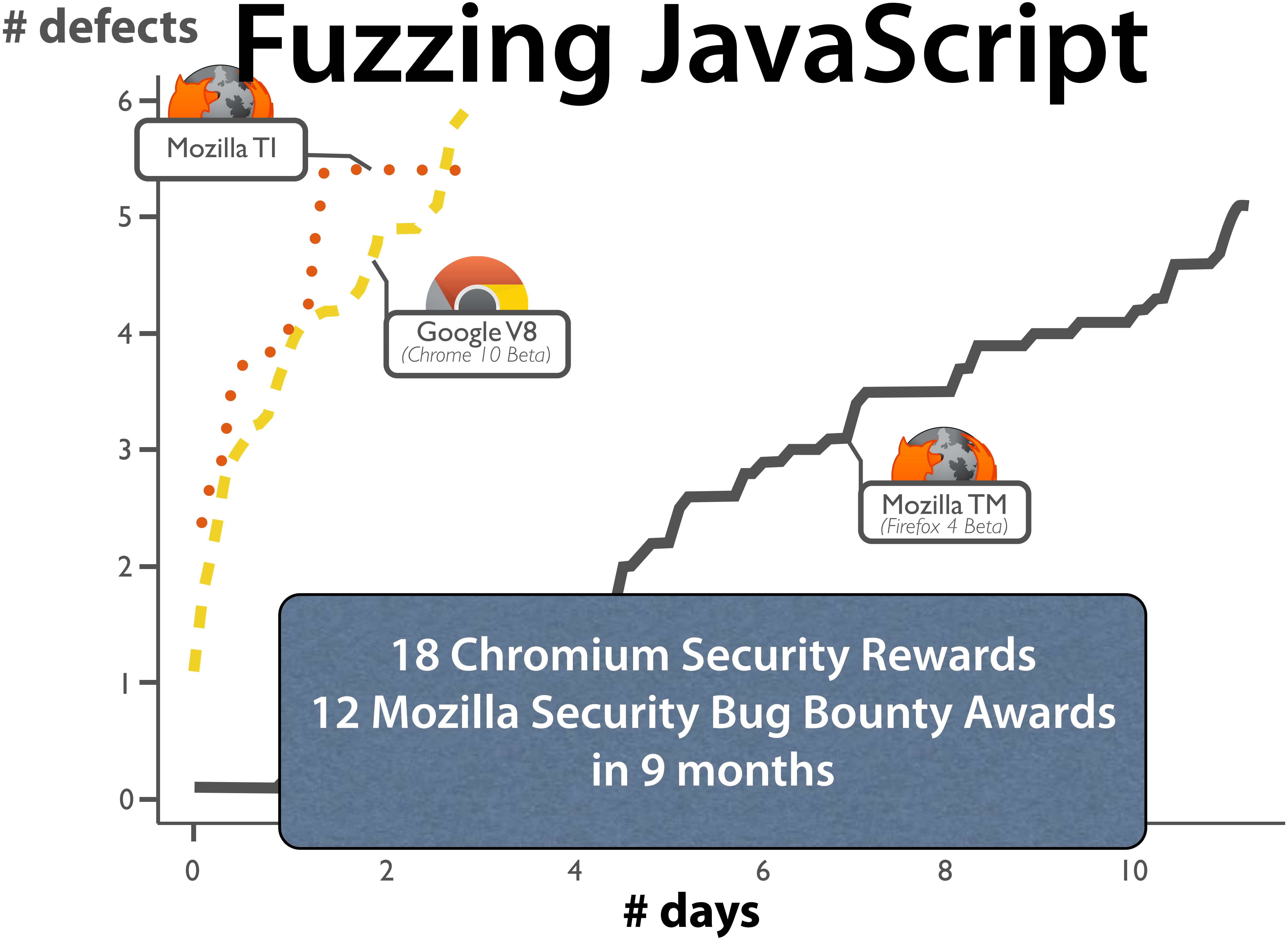
| *CaseGuards* *CaseGuard*

CaseGuard ⇒

A Generated Input

```
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);
```

Fuzzing JavaScript



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

Grammar Encoding

```
#!/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"]
}
```

grammar-fuzz.py

- Want to encode a *grammar* to produce arithmetic expressions as *strings*
- $START expands into $EXPR, which can expand into $TERM, $TERM + $TERM, etc.

```
#!/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"],
#!/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"]
}
```

```
    "$EXPR":
        ["$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 – $TERM
–$FACTOR + $TERM * –$FACTOR – $TERM
––$FACTOR + $TERM * –$FACTOR – $TERM
––$FACTOR + $FACTOR * –$FACTOR – $TERM
–––$FACTOR + $FACTOR * –$FACTOR – $TERM
–––$FACTOR + $FACTOR * –$FACTOR – $FACTOR
–––+$FACTOR + $FACTOR * –$FACTOR – $FACTOR
–––+–$F
–––+–$I
–––+–$D
–––+–2
–––+–2 + $INTEGER.$INTEGER * –$FACTOR – $FACTOR
–––+–2 + $INTEGER.$INTEGER * –+$FACTOR – $FACTOR
–––+–2 + $INTEGER.$INTEGER * –+$INTEGER – $FACTOR
–––+–2 + $DIGIT.$INTEGER * –+$INTEGER – $FACTOR
–––+–2 + 3.$INTEGER * –+$INTEGER – $FACTOR
–––+–2 + 3.$INTEGER * –+$INTEGER – +$FACTOR
–––+–2 + 3.$INTEGER * –+$INTEGER – +$INTEGER
–––+–2 + 3.$DIGIT * –+$INTEGER – +$INTEGER
–––+–2 + 3.5 * –+$INTEGER – +$INTEGER
–––+–2 + 3.5 * –+$DIGIT – +$INTEGER
–––+–2 + 3.5 * –+1 – +$INTEGER
–––+–2 + 3.5 * –+1 – +$DIGIT
–––+–2 + 3.5 * –+1 – +5
```

–––+–2 + 3.5 * –+1 – +5

Grammar Coverage

- Idea: Want to *cover* as many rules as possible
- Approach: Save that expansion has been applied (*covered*)
- Prefer *uncovered* over *covered* expansions

Grammar Coverage

$START

```
$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
```

Grammar Coverage

$EXPR

```
$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
```

Grammar Coverage

```
$EXPR + $TERM
```

```
$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
```

Grammar Coverage

```
$EXPR + $FACTOR
```

```
$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
```

Grammar Coverage

`$TERM + $FACTOR`

```
$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
```

Grammar Coverage

```
$TERM * $FACTOR + $FACTOR
```

```
  $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
```

Grammar Coverage

```
$TERM * $INTEGER + $FACTOR
```

```
$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
```

Grammar Coverage

`$TERM * $DIGIT + $FACTOR`

```
$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
```

Grammar Coverage

```
$TERM * 2 + $FACTOR
```

```
$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
```

Grammar Coverage

```
$TERM * 2 + $INTEGER.$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
```

Grammar Coverage

```
$TERM * 2 + $INTEGER$DIGIT.$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
```

Grammar Coverage

```
$TERM * 2 + $DIGIT$DIGIT.$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
```

Grammar Coverage

```
$TERM * 2 + 5$DIGIT.$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
```

Grammar Coverage

```
$TERM * 2 + 56.$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
```

Grammar Coverage

```
-8 / +7 * 2 + 56.9 ✅
```

```
$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✅
```

Tracking Grammar Coverage

- Track coverage during production
- Mark used productions as "covered"
- Prefer uncovered productions over covered ones
- Left as exercise to the reader 😀

Tracking Code Coverage

- You can also track coverage in Python
- Associate productions with coverage
- Prefer productions that lead to uncovered code
- Tracking coverage is very easy in Python!

Tracking Executions

```python
# Now, some dynamic analysis
def traceit(frame, event, arg):
    if event == "line":
        lineno = frame.f_lineno
        print("Line", lineno, frame.f_locals)
    return traceit

sys.settrace(traceit)
```

Course 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

Course Contents

Fuzzing 101

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

Grammar-Based Fuzzing

Structured fuzzing techniques
using *grammars* and models

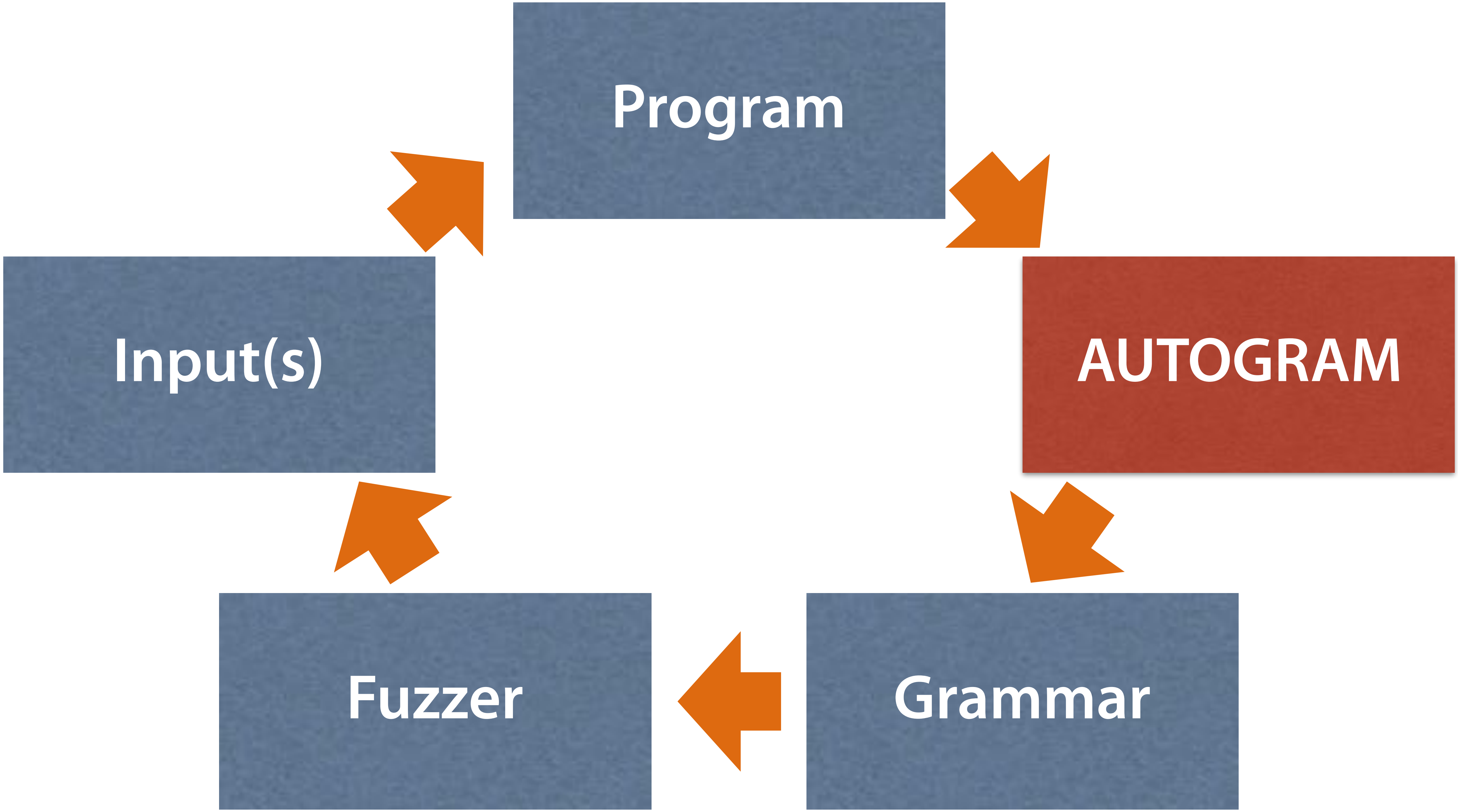
Inferring Grammars

Inferring input grammars
so you can fuzz arbitrary programs

Creating Grammars

```
URL ::= PR                                  Y] ['#' REF]
AUTHORITY
PROTOCOL :
USERINFO :
HOST ::= /
PORT ::= /
PATH ::= /
QUERY ::=
REF ::= /
```

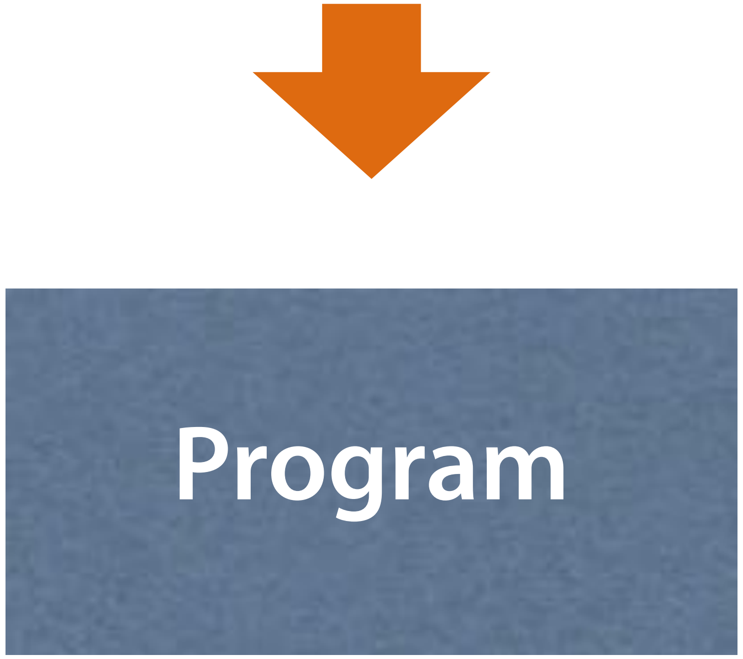
Learning Grammars



Höschele, 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`

`http`	– protocol
`www.google.com`	– host name
`80`	– port
`user pass`	– 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`

`http`	– protocol
`www.google.com`	– host name
`80`	– port
`user pass`	– login
`path`	– page request
`:// : @ : /`	– terminals

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
`:// : @ : /`	– terminals

processed in *different functions*

stored in *different variables*

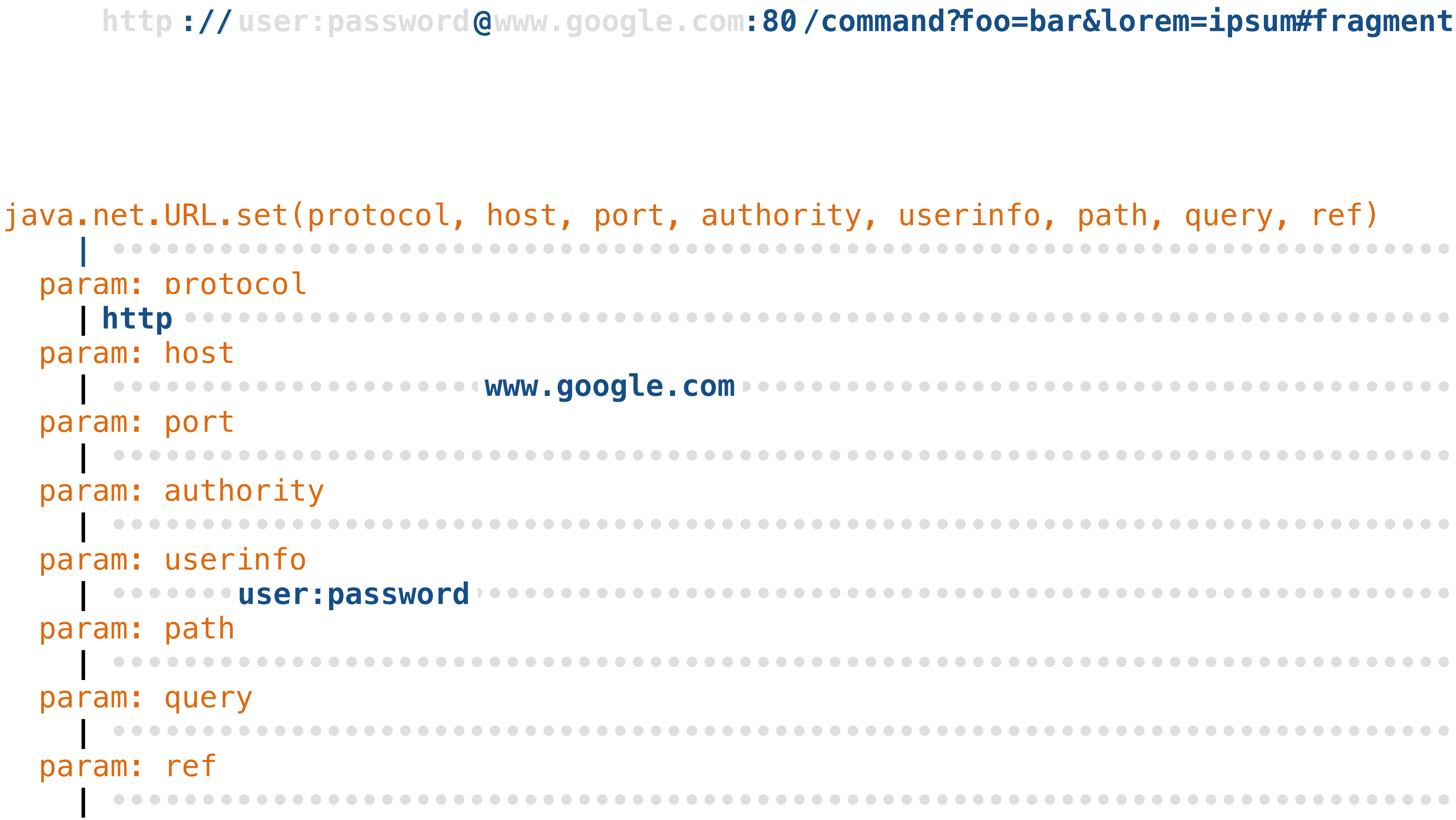
http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment

```
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
```

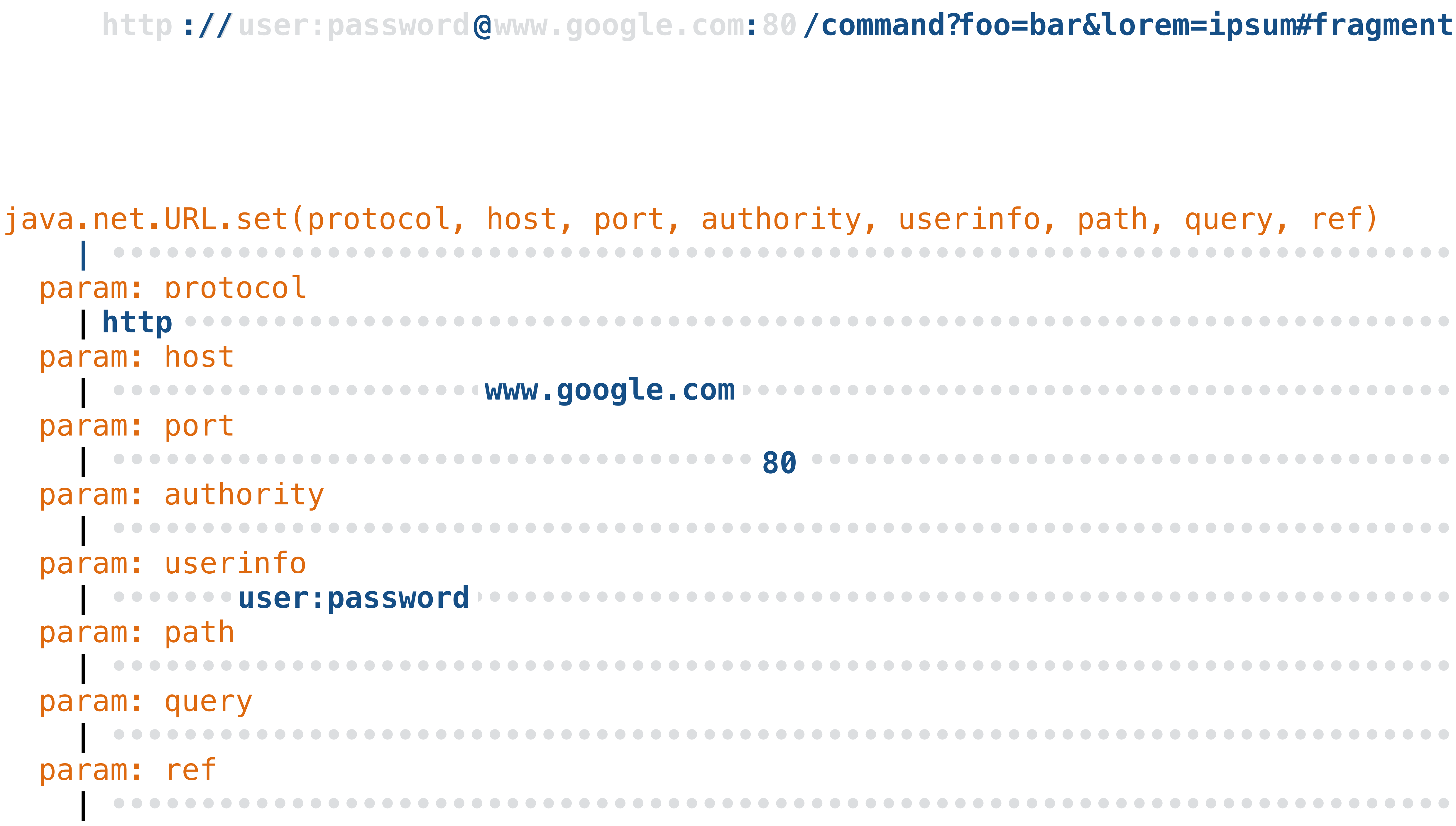
```
    http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment


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

```
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
    |
  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

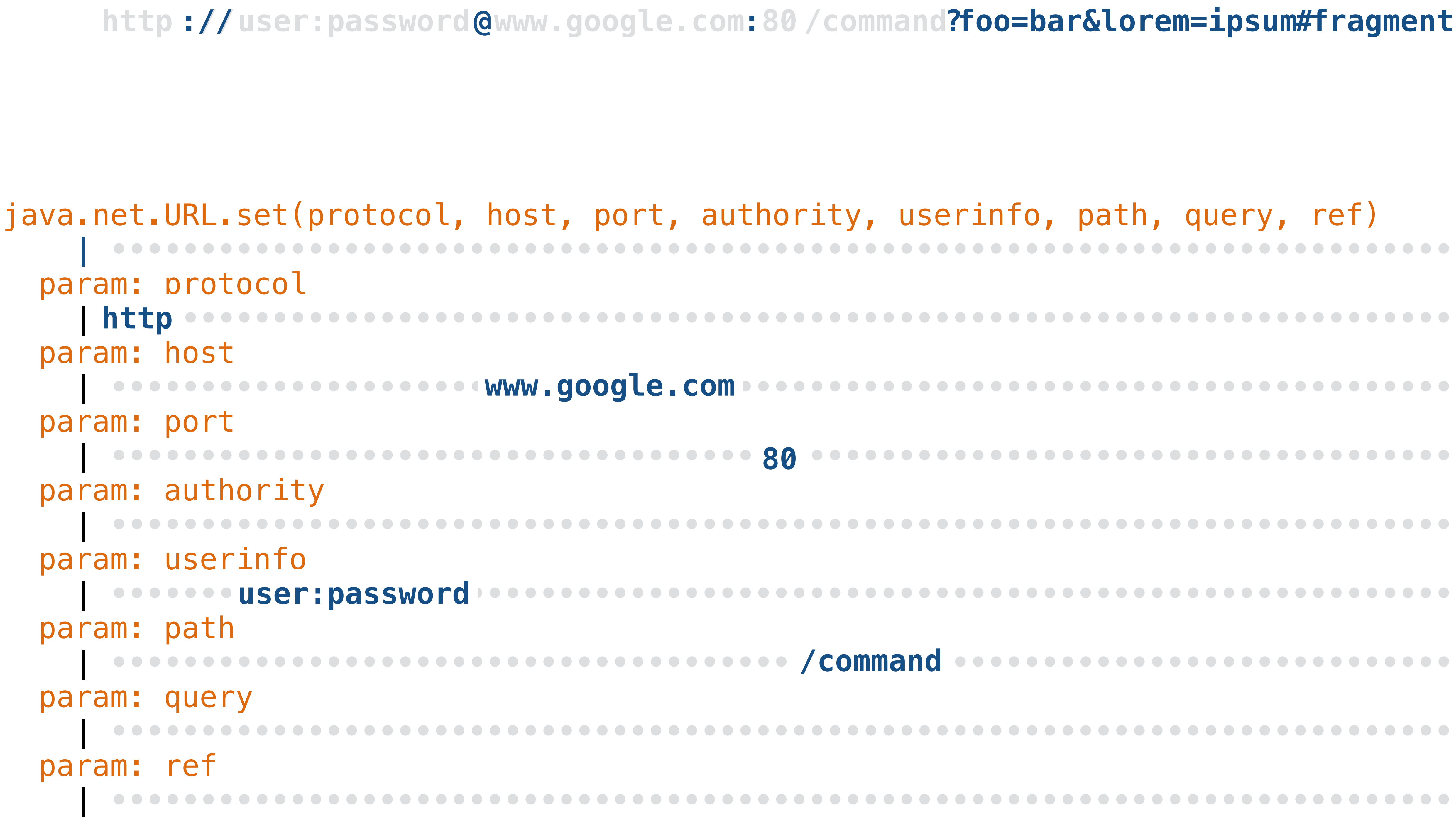
param: userinfo

user:password

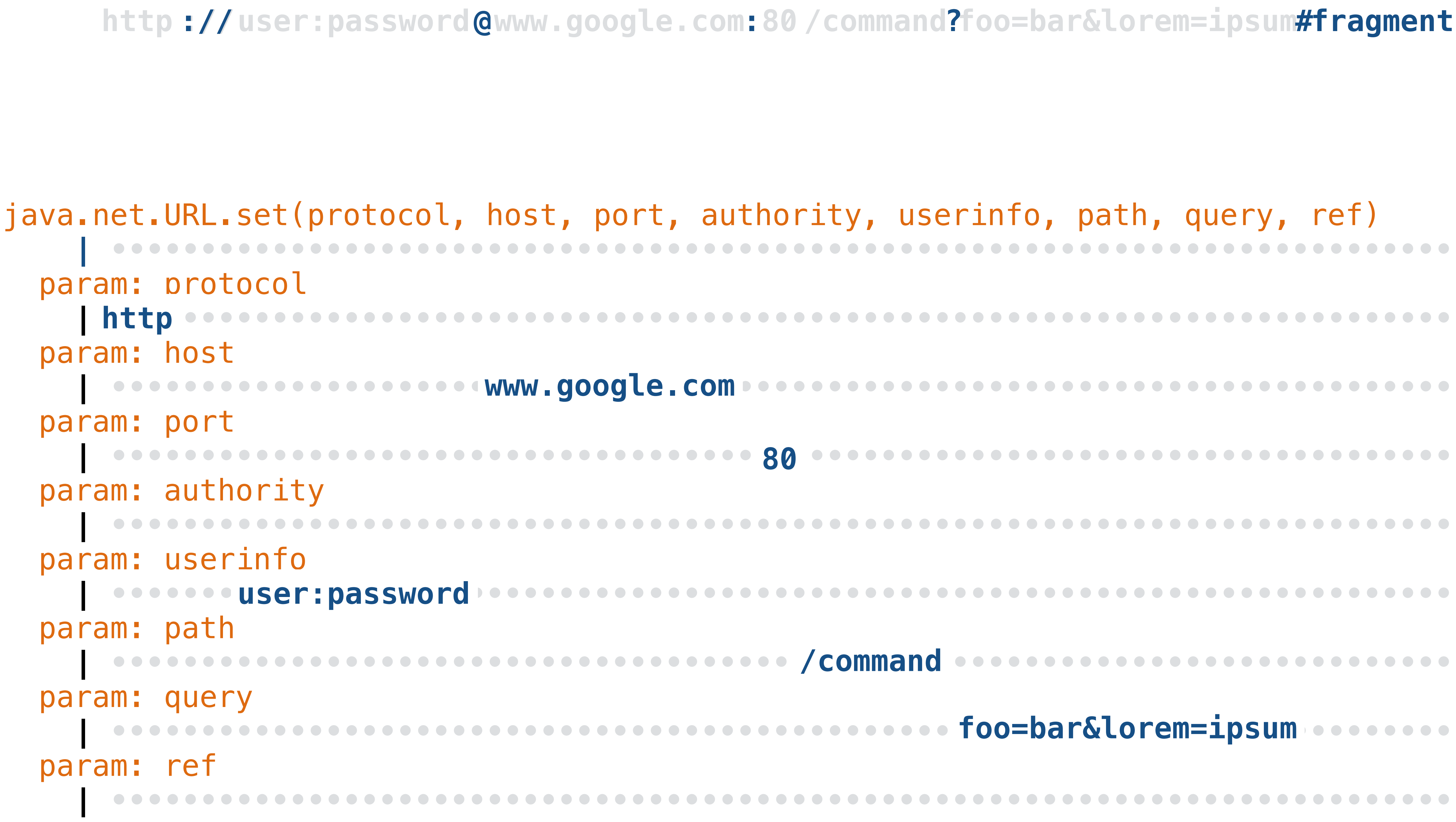
param: path

param: query

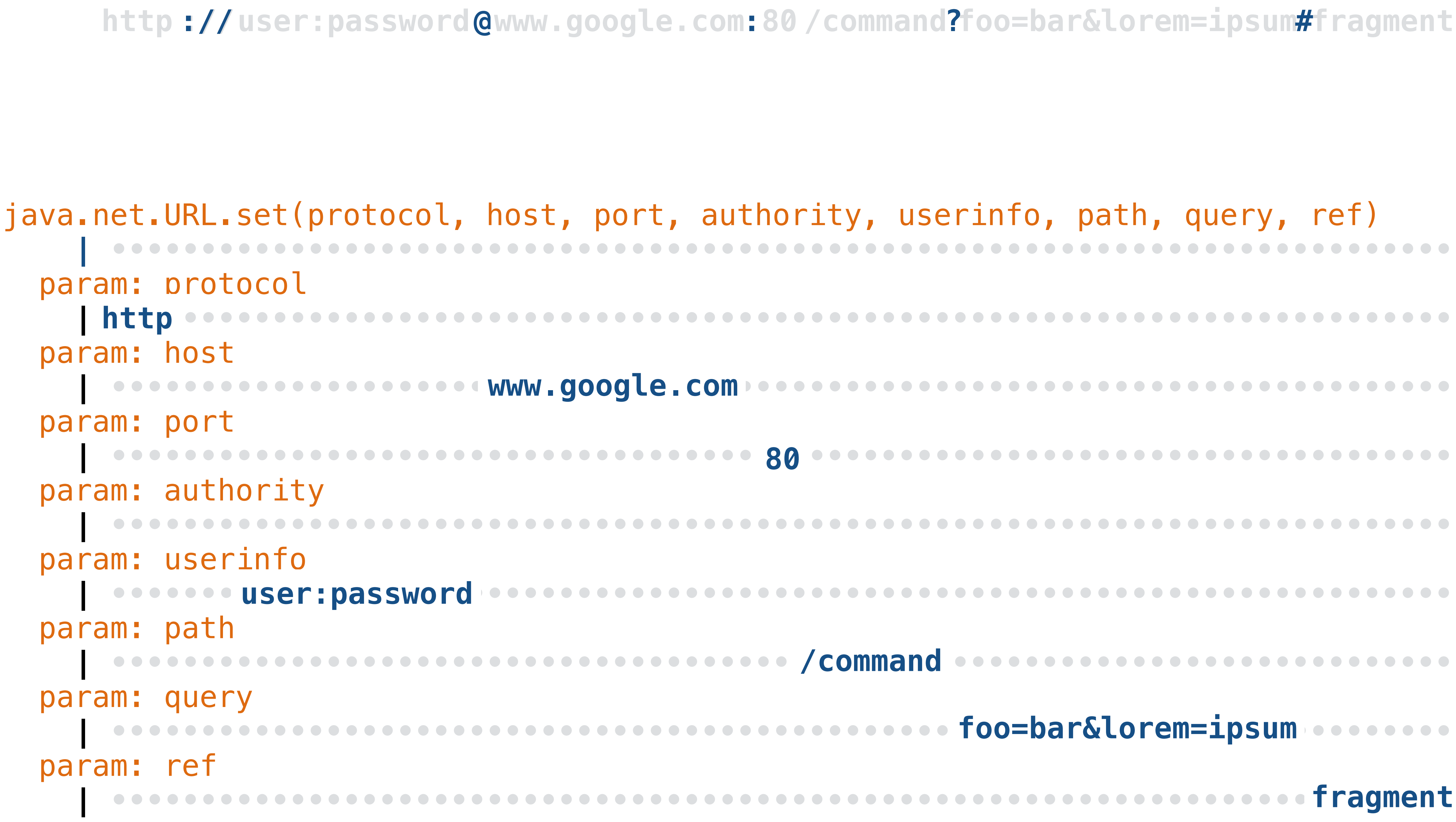
param: ref



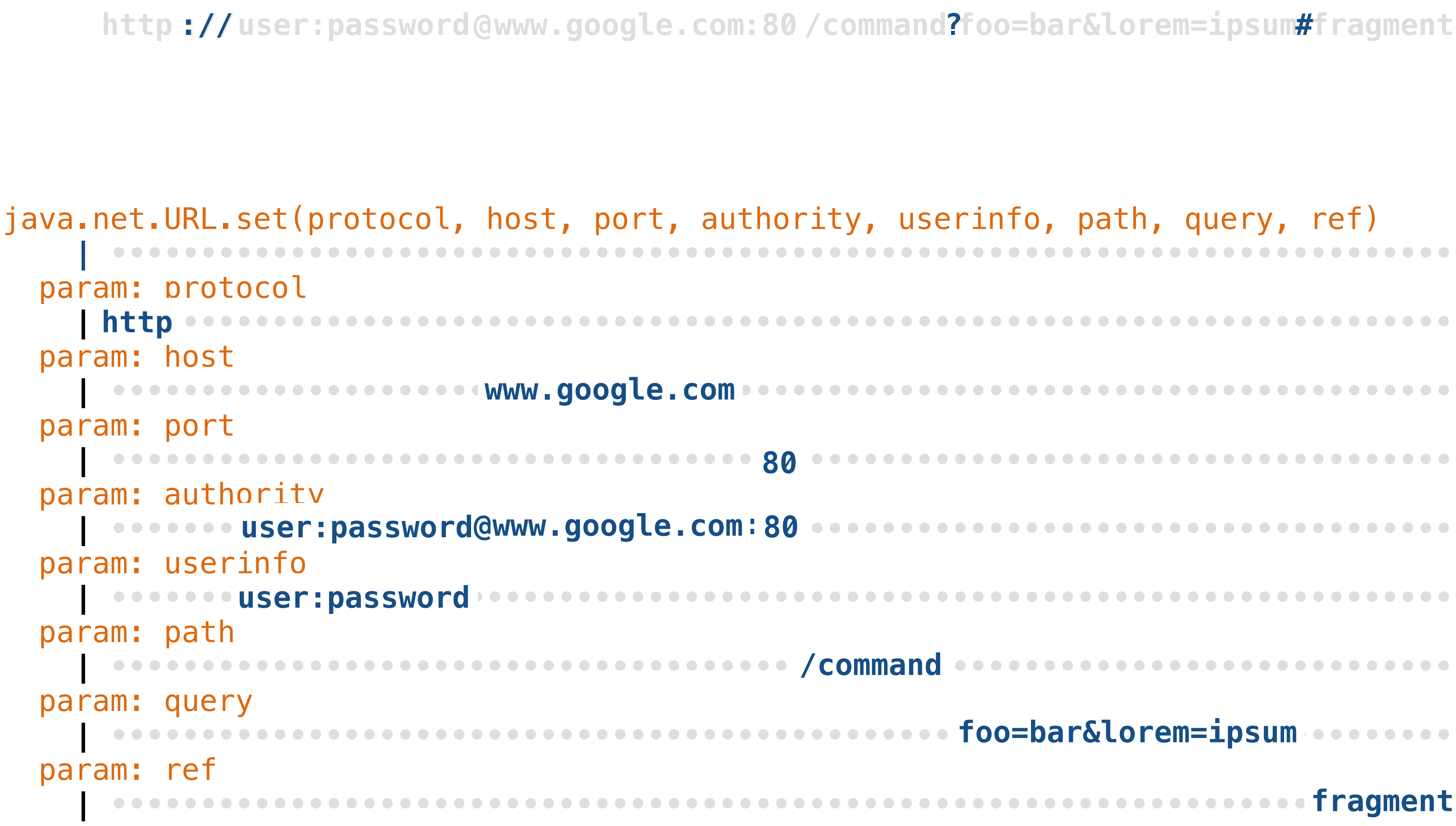
```
     http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment


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
    |
  param: userinfo
    |        user:password
  param: path
    |                                        /command
  param: query
    |                                                 foo=bar&lorem=ipsum
  param: ref
    |
```

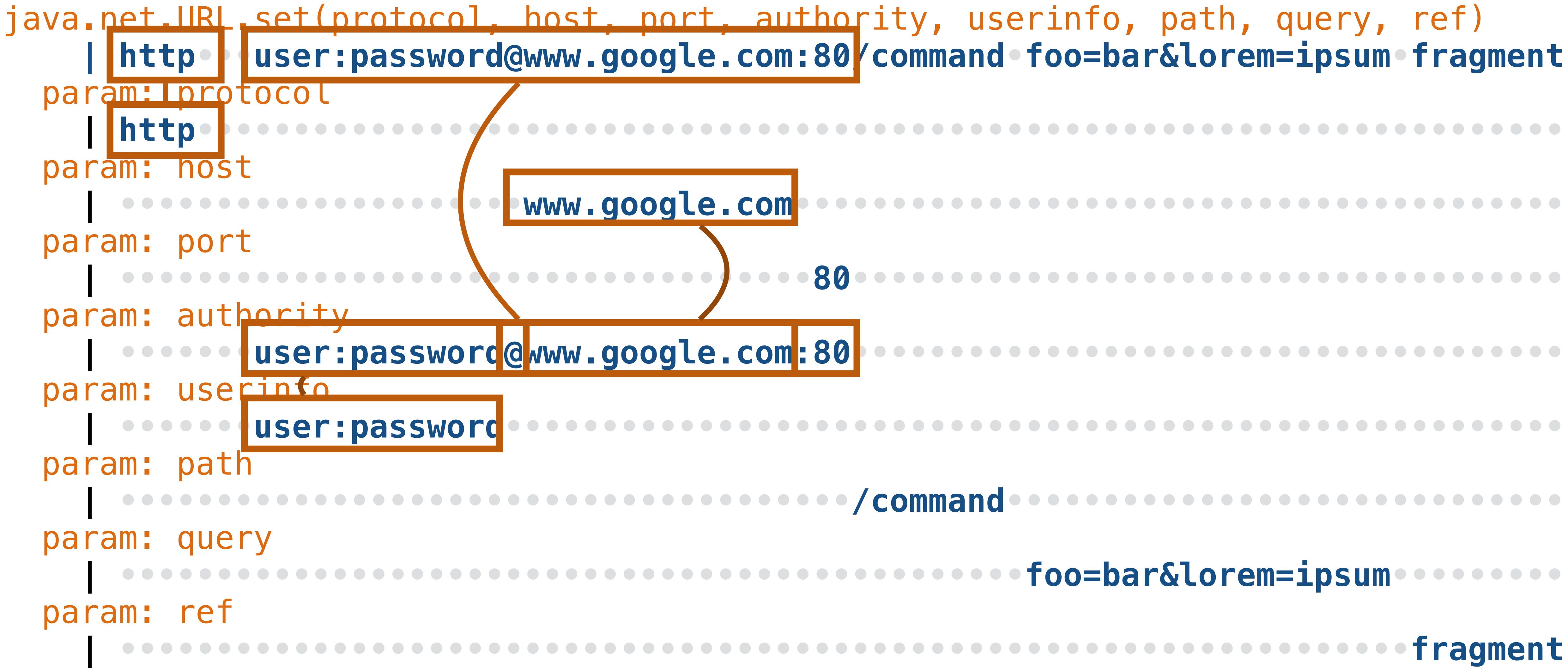


```
     http://user:password@www.google.com:80/command?foo=bar&lorem=ipsum#fragment


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
    |
  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   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
```



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
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 → *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 → *value*

Tracking

```python
# 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)
```

Grammar Expansions

```
# 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]
    # 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]
            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*

Merging Grammars

```
# 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
```

$$\cup$$

```
'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
```

↓

```
INI ::= LINE+
LINE ::= SECTION_LINE '\r'
       | OPTION_LINE  ['\r']
SECTION_LINE ::= '[' KEY ']'
OPTION_LINE ::= KEY ' = ' VALUE
KEY ::= /[a-zA-Z]*/
VALUE ::= /[a-zA-Z0-9\/]/
```

JSON Input

```
{
 "v": true,
 "x": 25,
 "y": -36,
 …
}
```

```
JSON ::= VALUE
VALUE ::= JSONOBJECT | ARRAY | STRINGVALUE |
          TRUE | FALSE | NULL | NUMBER
TRUE ::= ’true’
FALSE ::= ’false’
NULL ::= ’null’
NUMBER ::= [’-’] /[0-9]+/
STRINGVALUE ::= ’"’ INTERNALSTRING ’"’
INTERNALSTRING ::= /[a-zA-Z0-9 ]+/
ARRAY ::= ’[’
 [VALUE [’,’ VALUE]+]
’]’
JSONOBJECT ::= ’{’
 [STRINGVALUE ’:’ VALUE
  [’,’ STRINGVALUE ’:’ VALUE]
  +]
'}'
```

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

Fuzzing File Formats

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.

Contents [hide]

Course 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

Course 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*!

N. Havrikov

M. Höschele

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!

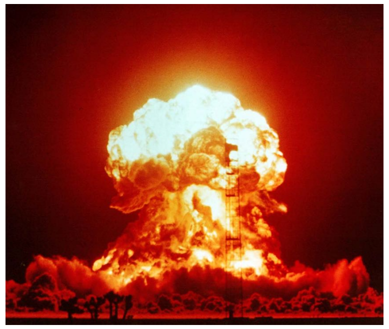
Christian Holler

CISPA Saarbrücken

© av-a Veauthier Architekten

Grammar-Based Fuzzing

```
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);
```



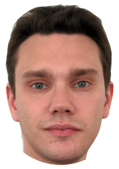
Grammar Inference

```
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'
```

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*!



N. Havrikov

M. Höschele

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!