’T ain’t enough to fuzz

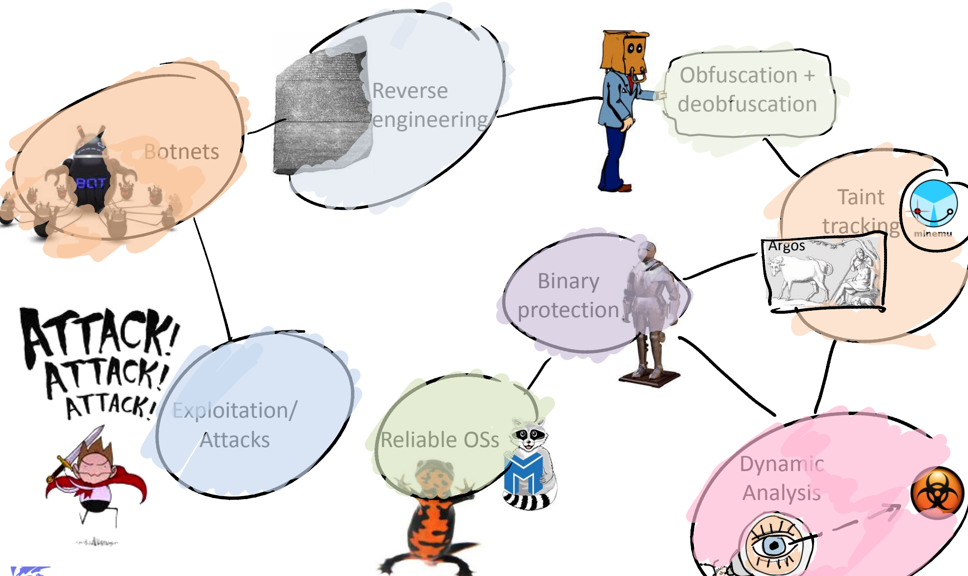
Herbert Bos
VU University Amsterdam

Heavy lifting

- Istvan Haller
- Asia Slowinska
- Erik Bosman
- Victor van der Veen

Some things we do

GreHack

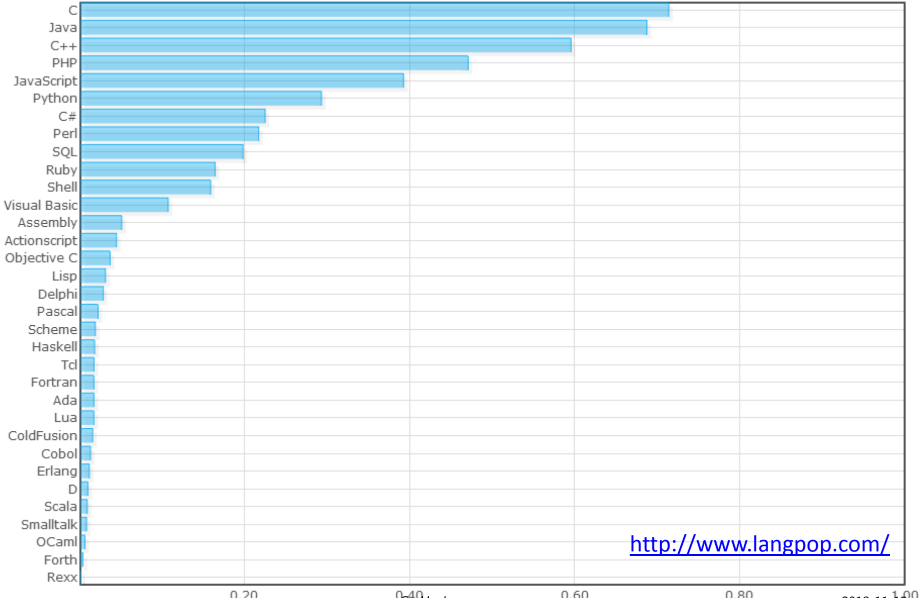


Today:

Buffer Overflows

The most popular language in the world

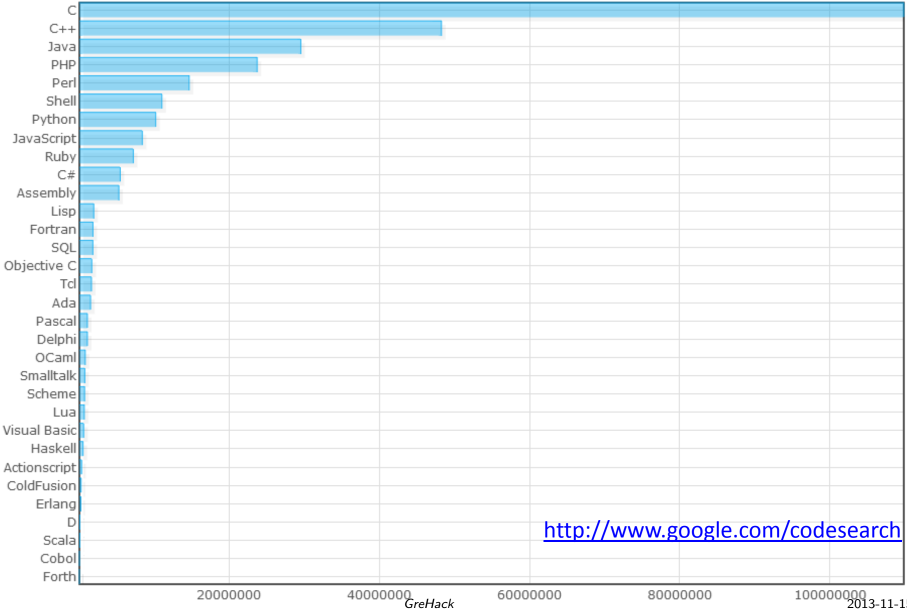
GreHack



http://www.langpop.com/

The most popular language in the world

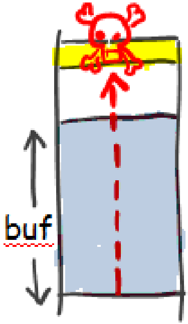
GreHack



http://www.google.com/codesearch

Buffer overflows

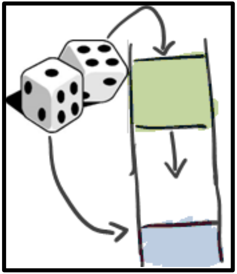
- Perpetual top-3 threat
  - SANS CWE Top 25 Most dangerous programming errors
- Most drive-by-downloads
  - infect browser, download malware



Many defensive measures

- NX bit / DEP / W⊕X
- Canaries and Cookies
- ASLR

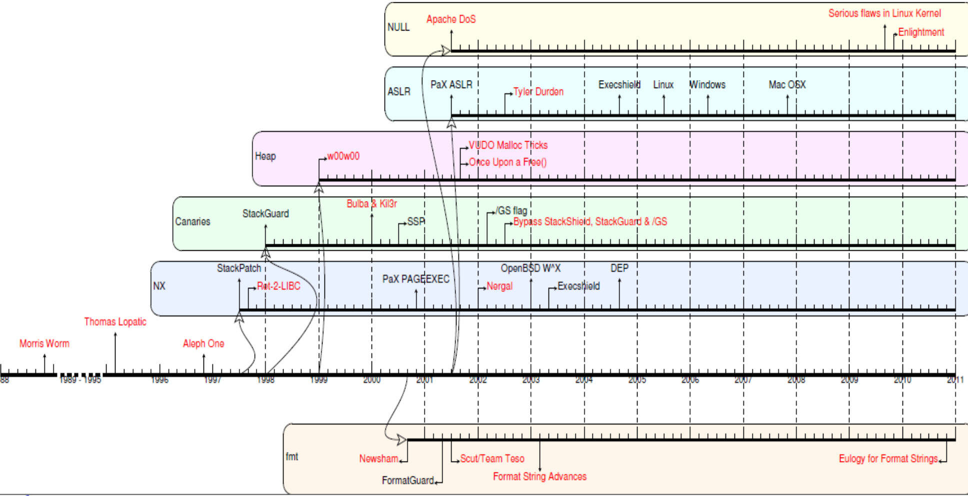




Still they come

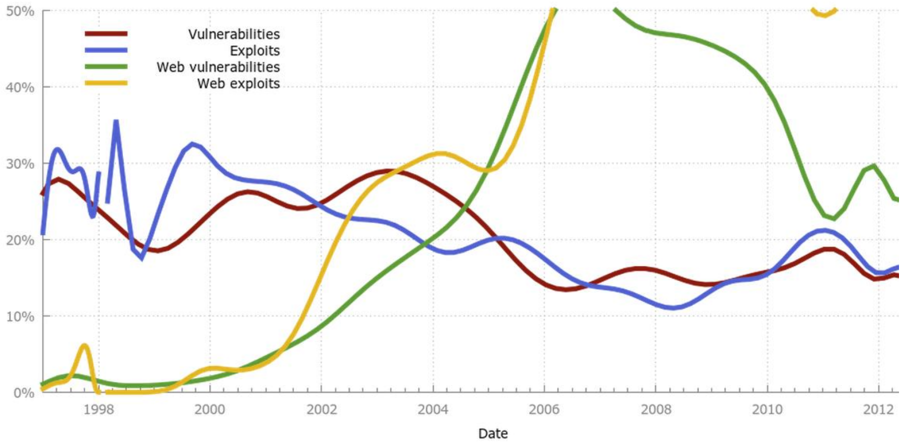
Evolution at work

"Memory Errors: the Past, the Present and the Future" [RAID'12]



Vulnerabilities and exploits

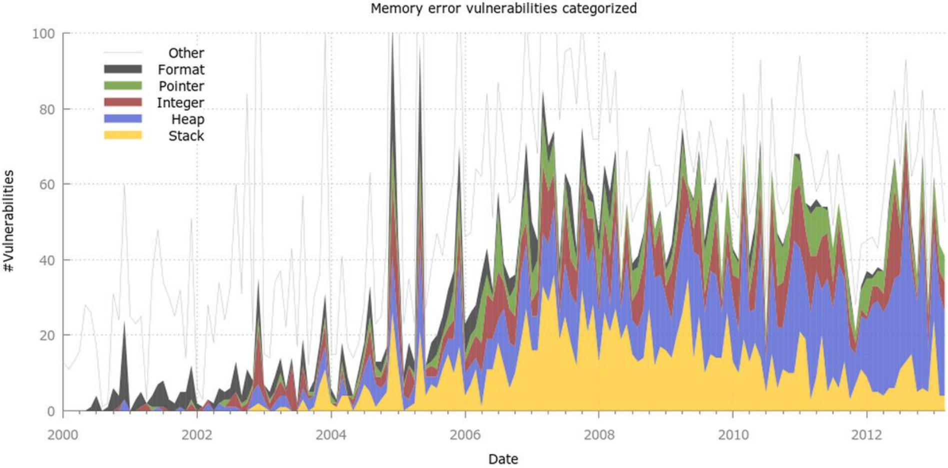
(as percentage of total)



http://vvdveen.com/graphs/webpage.html

Nature of attacks

(stack-based overflows are getting rarer)



http://vvdveen.com/graphs/webpage.html

wouldn't it be nice

if we found them

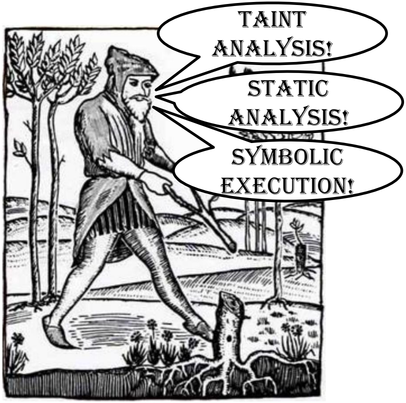
automatically

before release

Testing

Dowsing

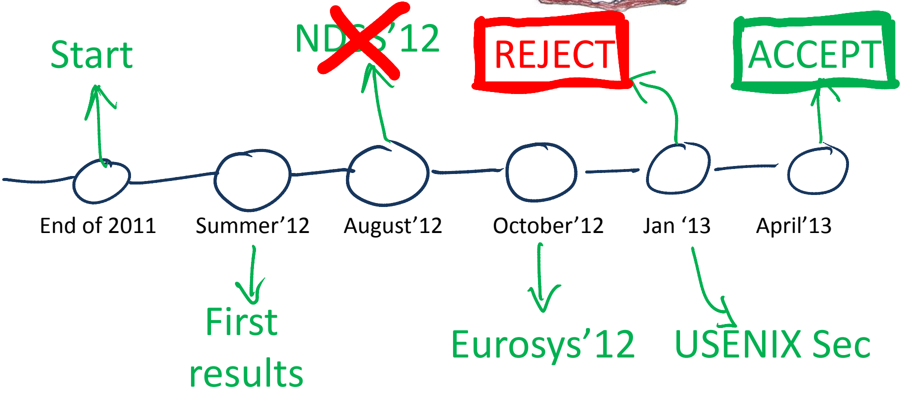
A Guided Fuzzer to Find Vulnerabilities



Dowsing is a type of divination used to find ground water buried treasure, rare gemstones, and now also bugs...

GreHack

Timeline



Where's the fire?

- Buffer overflows are a top 3 threat!
  - Triggered under rare conditions
- Applications grow rapidly
  - Automated testing doesn't scale!

Security testing today

Symbolic execution

- Example: let's model the speed of a car

Concrete values	Symbolic values
115 km/h	$100 \leq v \leq 120$ km/h
115 km/h	$0 <= v <= 120$ km/h
250km/h	$v >= 0$ km/h

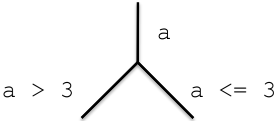
Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass?(0);
}
```

a

Symbolic execution

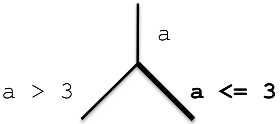
```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass?(0);
}
```



GreHack

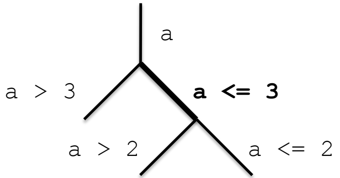
Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass?(0);
}
```



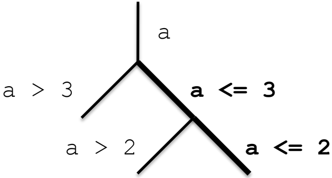
Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass?(0);
}
```



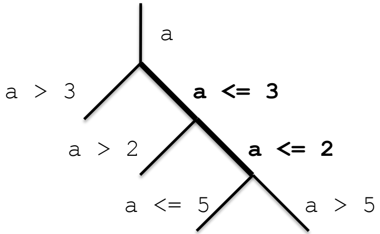
Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass?(0);
}
```



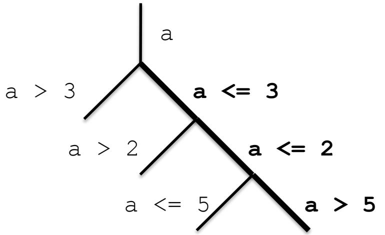
Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass?(0);
}
```



Symbolic execution

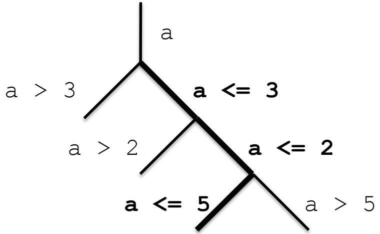
```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass?(0);
}
```



(a<=3)&&(a<=2)&&(a>5) ✗

Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                assert(0);
}
```



(a<=3)&&(a<=2)&&(a<=5)

(a<=2)

Symbolic execution

- Does not scale!
  - The number of states grows exponentially, so the analysis of a complex program can take ages!
  - E.g., nginx vulnerability not found within 8 hours

But we don't want to test the entire program

Only the buggy bits!

Surely, bugs can be anywhere!

- Can they?
- What do we need for a buffer overflow?
  - Buffer
  - Accesses to that buffer
  - Loop
- We can look for these properties *a priori*!

Moreover…

- All loops are created equal, but some loops are more equal than others
  - Complex code is buggier than simple code
  - …

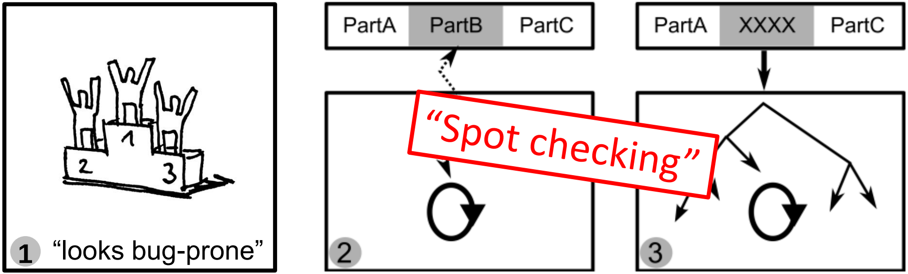
Buffer underrun in nginx

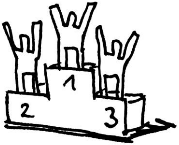
```
while (p <= r->uri_end)
    switch (state)
    case sw_usual: *u++ = ch; ...
    case sw_slash: *u++ = ch; ...
    ...
    case sw_dot: *u++ = ch; ...
        if (ch == '/') u--; ...
    case sw_dot_dot: *u++ = ch; ...
        if (ch == '/') u -= 4; ...
    ...
```

400 lines of code that make your head hurt

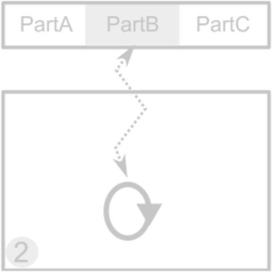
Idea: dowse for vulnerabilities

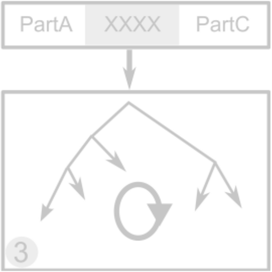
- Don’t try to verify all inputs
  - Focus the search for bugs on small and “potentially suspicious” code fragments

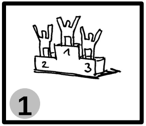




1 “looks bug-prone”



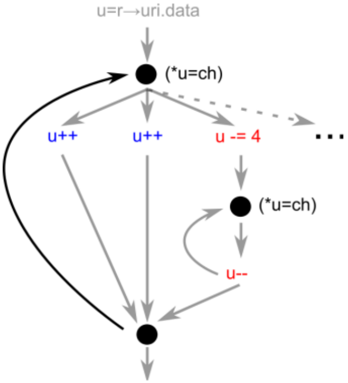


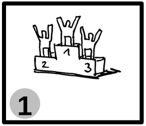


Identify places likely to have bugs

Buffer overflows in software

- Requirements:
  - An array
  - A pointer accessing the array
  - In a loop
- Find statically
  - Hundreds – thousands of loops
- Our strategy:
  - Analyze data flow graph
  - Rank based on complexity



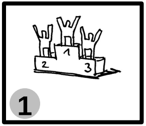


GreHack

How do we rank?

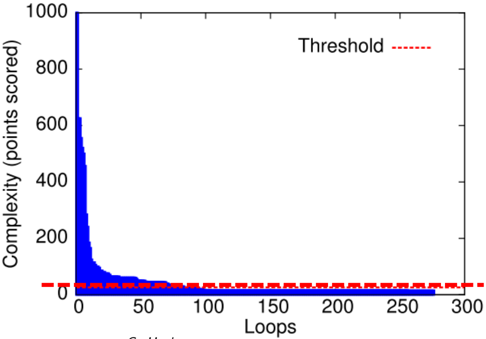
- We score based on
  - Instructions
  - Different constants
  - Pointer casts
  - ….

Instructions	Points
Array index manipulations	
Basic index arithmetic instructions, i.e., addition, and subtraction	5
Other index arithmetic instructions, e.g., division, multiplication, shift, and xor	10
Different constant values	10
Constants involved in accessing fields of structures	0
Numerical values determined outside the loop	30
Non-inlined functions returning non-pointer values	500
Data movement instructions	0
Pointer manipulations	
Loading a pointer calculated outside the loop, e.g., an operation retrieving the base pointer of an object	0
`GetElemPtr` – an LLVM instruction that computes a new pointer from a base and offset(s)	5
Pointer cast operations, i.e., `PtrToInt` and `IntToPtr`	100

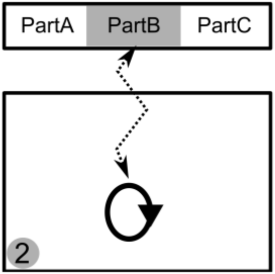


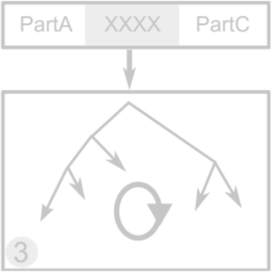
Does that work?!

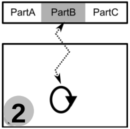
- Consider nginx…
  - 70% of loops have minimal complexity
  - Example loop is in the top 5%




2
3
1
1 "looks buggy"


PartA
PartB
PartC
2

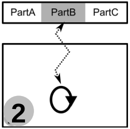

PartA
XXXX
PartC
3



Input tracking

Peter and Dorothy Denning: tracking information flows since the 1970s!

- Aim:
  - Infer relationships between inputs and candidates
  - Taint tracking

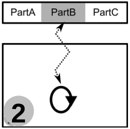


Input tracking

Example: nginx HTTP request

Long input with multiple tokens.

GET /long/path/file HTTP/1.1
Host: thisisthehost.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 1337



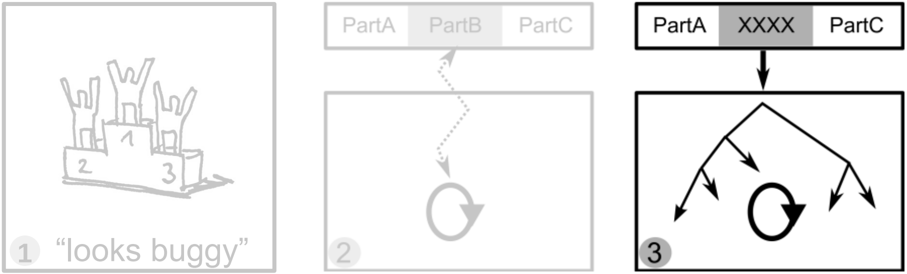
Input tracking

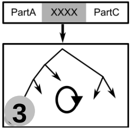
Example: nginx HTTP request

Only small part influences given loop

GET /long/path/file HTTP/1.1
Host: thisisthehost.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 1337

→ Make only these bytes symbolic


1
"looks buggy"
PartA
PartB
PartC
2
PartA
XXXX
PartC
3

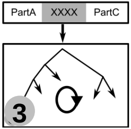


Symbolic execution

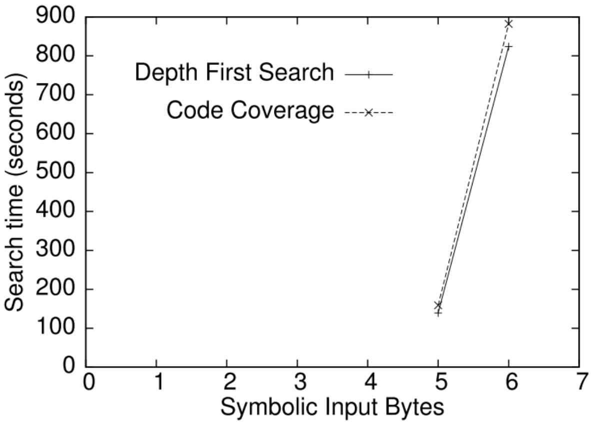
Now possible?

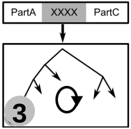
Not quite, but getting close

More tricks are in the paper [USENIX SEC’13]

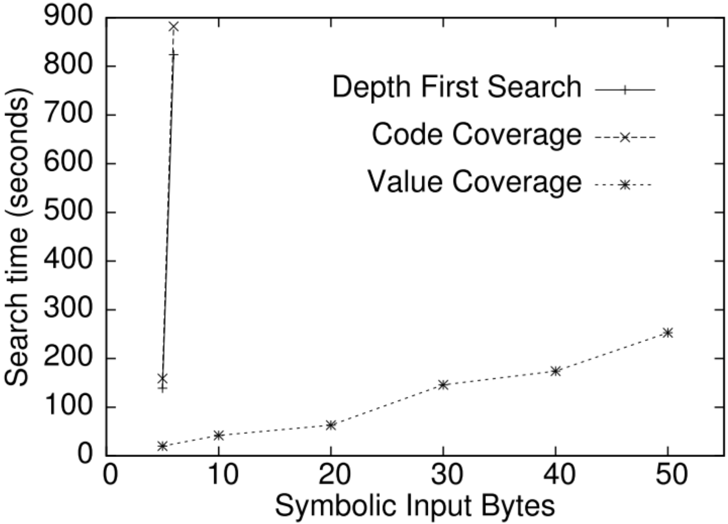


Symbolic execution





Our approach



Results

Program	Vulnerability	*Dowser*	Symbolic input
nginx 0.6.32	CVE-2009-2629 heap underflow	253 sec	URI field 50 bytes
ffmpeg 0.5	UNKNOWN heap overread	48 sec	Huffman table 224 bytes
inspircd 1.1.22	CVE-2012-1836 heap overflow	32 sec	DNS response 301 bytes
poppler 0.15.0	UNKNOWN heap overread	14 sec	JPEG image 1024 bytes
poppler 0.15.0	CVE-2010-3704 heap overflow	762 sec	Embedded font 1024 bytes
libexif 0.6.20	CVE-2012-2841 heap overflow	652 sec	EXIF tag/length 1024 + 4 bytes
libexif 0.6.20	CVE-2012-2840 off-by-one error	347 sec	EXIF tag/length 1024 + 4 bytes
libexif 0.6.20	CVE-2012-2813 heap overflow	277 sec	EXIF tag/length 1024 + 4 bytes
snort 2.4.0	CVE-2005-3252 stack overflow	617 sec	UDP packet 1100 bytes

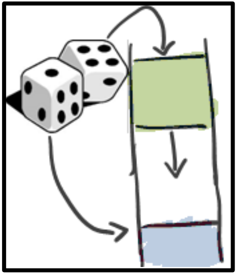
So we found a buffer overflow

Now what?

How to make use of it?

- DEP makes direct execution of shellcode unlikely
- Instead: code reuse
  - Return to libc
  - ROP





NEW!

SIGRETURN ORIENTED PROGRAMMING

Powerlifting

- Erik Bosman

[deliberately left blank]

The SROP material has not been made public yet

Conclusions

- Memory corruption are here to stay
  - Good hunting ground for research topics
- Only scratching the surface of fuzzing
  - Dowsing looks promising
- Interesting to look at new defenses
  - CFI anyone?
- Shellcode, ROP, JOP, …
  - Now SROP → not the final word