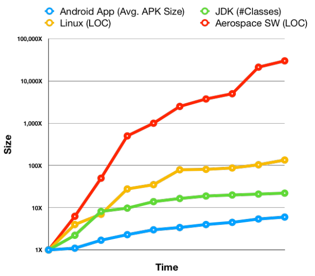
Chisel: A System for Debloating C/C++ Programs

[to be completed]   , Mayur Naik

Motivation

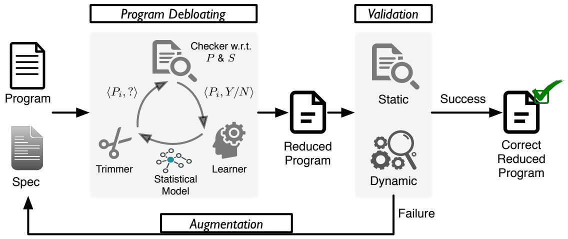
> “Perfection is achieved not when there is nothing left to add, but when there is nothing left to take away.”
>
> — Antoine de Saint-Exupéry

Growth of Software Complexity



Consequence: degraded performance and expanded attack surface

Solution: late-stage customization by removing redundant functionalities



Problem Statement

Given a program *P* to be minimized and a property test function *S*, find a 1-minimal program *P’* that is a subset of *P* and satisfies the property.

The property test function can be expensive to invoke.

Method

Desired Properties

Minimality: trim code as aggressively as possible w.r.t the spec

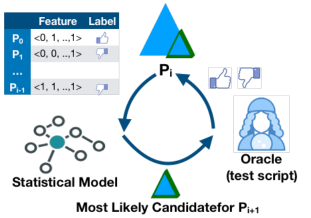
Efficiency: find the minimized program in a scalable manner

Robustness: avoid introducing new vulnerabilities

Naturalness: produce maintainable and extensible code

Generality: handle a wide variety of programs and specs

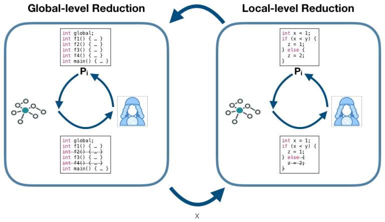
Learning-Guided Delta Debugging



Learn a policy via reinforcement learning
Guide the search based on the prediction of the learned policy

Discard nonsensical programs upfront using known ***hard-rules***

Alternate between structural levels of the program to enhance ***scalability***



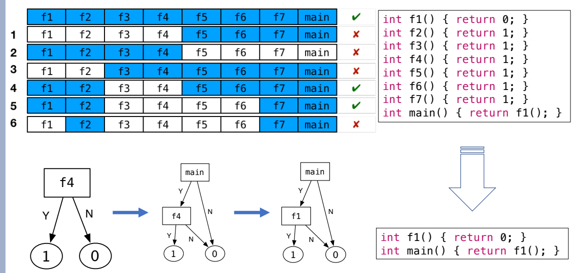
```
int global;
int f1() { … }
int f2() { … }
int f3() { … }
int f4() { … }
int main() { … }
```

```
int global;
int f1() { … }
~~int f2() { … }~~
int f3() { … }
~~int f4() { … }~~
int main() { … }
```

```
int x = 1;
if (x < y) {
  z = 1;
} else {
  z = 2;
}
```

```
int x = 1;
if (x < y) {
  z = 1;
} ~~else {~~
  ~~z = 2;~~
}
```

Example of a Guided Search



	f1	f2	f3	f4	f5	f6	f7	main	✔
1	f1	f2	f3	f4	f5	f6	f7	main	✘
2	f1	f2	f3	f4	f5	f6	f7	main	✘
3	f1	f2	f3	f4	f5	f6	f7	main	✘
4	f1	f2	f3	f4	f5	f6	f7	main	✔
5	f1	f2	f3	f4	f5	f6	f7	main	✔
6	f1	f2	f3	f4	f5	f6	f7	main	✘

```
int f1() { return 0; }
int f2() { return 1; }
int f3() { return 1; }
int f4() { return 1; }
int f5() { return 1; }
int f6() { return 1; }
int f7() { return 1; }
int main() { return f1(); }
```

```
int f1() { return 0; }
int main() { return f1(); }
```

Experiments

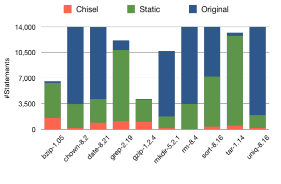
10 widely used UNIX utility programs

Each program has a known CVE

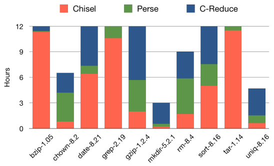
Only supporting command-line options as BusyBox

Code, benchmark, and docs: https://chisel.cis.upenn.edu

More Effective than State-of-the-art



Reduction Size



Reduction Time

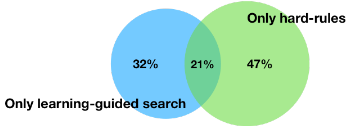
Security Hardening

		#Gadget			#Alarms		
Program	**CVE**	**Original**	**Reduced**		**Original**	**Reduced**	
bzip-1.05	✘	662	298	55X	1,991	33	98X
chown-8.2	✔	534	162	70X	47	1	98X
date-8.21	✔	479	233	51X	201	23	89X
grep-2.19	✔	1,065	411	61X	619	31	95X
gzip-1.2.4	✔	456	340	25X	326	128	61X
mkdir-5.2.1	✘	229	124	46X	43	2	95X
rm-8.4	✘	565	95	83X	48	0	100X
sort-8.16	✔	885	210	76X	673	5	99X
tar-1.14	✔	1,528	303	80X	1,290	19	99X
uniq-8.16	✘	349	109	69X	60	1	98X
Total		**6,752**	**2,285**	**66X**	**5,298**	**243**	**95X**

Reduced potential attack surface   Feasible manual inspection

Assessing the Effect of different Pieces

The significant performance improvement is a result of incorporation hard-rules as well as learning-guided search


Only hard-rules
Only learning-guided search