A Broad Comparative Evaluation of Software Debloating Tools

USENIX Security 2024

Michael D. Brown

15 August 2024

Why Evaluate Software Debloaters?

Software debloating is an emerging research area aiming to remove unnecessary code from programs to:

- **Improve performance**
- **Improve security posture (less code, less attack surface)**

However, evaluations of tools to date are limited in scope and use inconsistent sets of metrics.

This makes it hard for potential users to know what tools to use, what benefits to expect, and whether they are safe/effective.

Motivating Questions

We designed an evaluation for SotA debloating tools to answer:

1. **How can debloating tools be evaluated?**
   - What metrics should be used?
   - What benchmarks should be used?

1. **How well do these tools perform relative to each other?**

1. **What barriers to adoption exist for software debloaters?**

Some Background

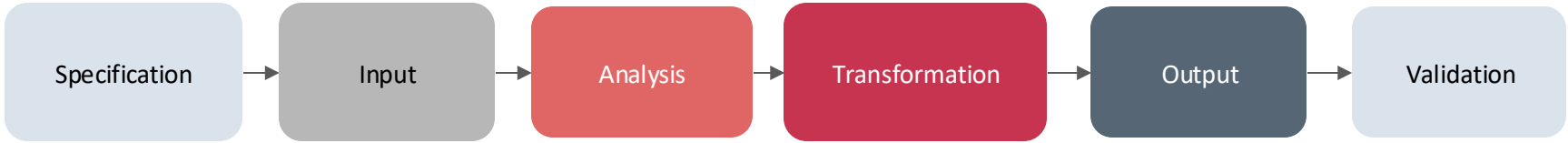
Survey of Debloating Techniques

There have been over 70 publications in the last 10 years for removing bloat in:

- Software (Source, Binary, IR)
- Containers
- OSes and their APIs
- Firmware
- Test cases
- Build dependencies
- And more

We focus on software for x86[_64] architectures

How Do Debloaters Work, Generally?



Types of Bloat

Two categories of software bloat:

Type I: Universally unnecessary for all intended uses

- E.g., Library code, API functions that are never called

Type II: Conditionally unnecessary depending on intended use

- E.g., Features a particularly user doesn't need, code for targeting multiple architectures

Types of Debloaters (Type I)

Static Library (SL)

- Target unnecessary library functions (dynamically loaded)
- Analyze call graph to find unnecessary library functions, then remove or blank them
- **Pros**: Low soundness risks, do not require specs
- **Cons:** Fragments shared libraries on system

Runtime

- Dynamic version of SL debloaters
- Uses reachability information at runtime to select, excise, or blank bloat library functions
- **Pros**: Avoids library fragmentation
- **Cons:** Very complex, significant overheads

Types of Debloaters (Type II)

Source to Source (S2S)

- Target unnecessary program features user doesn't need
- Analysis maps features to code, then removes code associated with unwanted features
- **Pros**: Targets richest program rep, compiler helps identify problems
- **Cons:** Can require exhaustive test cases, requires source code

Binary to Binary (B2B)

- Binary version of S2S debloaters
- Requires binary disassembly / decompilation / lifting
- **Pros**: Can debloat legacy binaries
- **Cons:** High risk of soundness issues, removing code is challenging (blanking is typical)

Types of Debloaters (Type I + II)

Compiler-Based Specializers (CBS)

- Can target multiple types of bloat
- User specifies one or more arguments as compile-time constants, use compiler to remove bloat as “dead code”
- **Pros**: Low soundness risks, specs are easy to generate
- **Cons:** Limited to aggressive debloating of CLI applications only

Debloater Metrics

30 different evaluation metrics found:

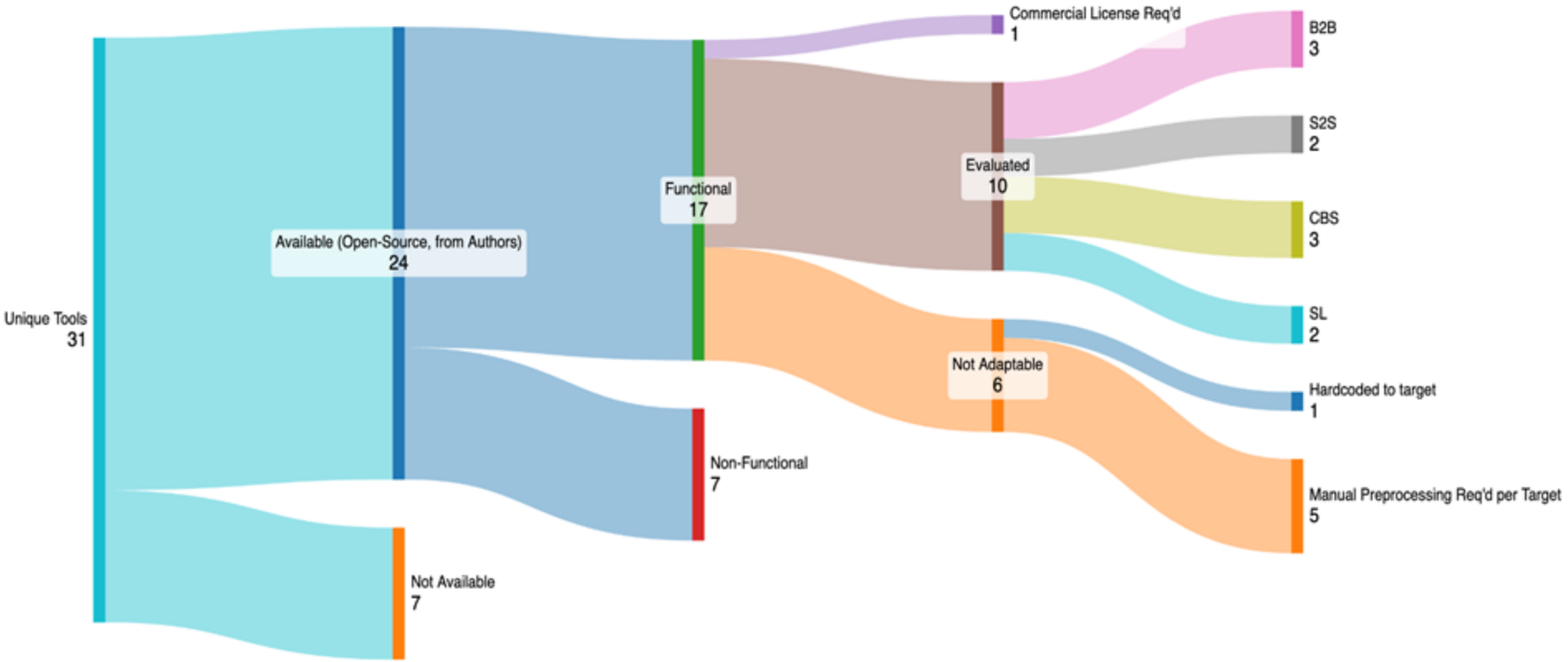
1. Performance: e.g., runtime, size, memory consumption

2. Correctness / Robustness: e.g., failures and crashes

3. Security Improvement: e.g., CVEs removal, code reusability

Evaluation

Tool Selection



Tool Selection

Table 1: Debloaters Selected for Evaluation

Year	Cite	Name	Category	Bloat Type	Granularity	Spec. Type	Open Source
2018	[31]	CHISEL	S2S	Type II	Line	Test Cases	•
2023	[31]	CHISEL-GT	S2S	Type II	Line	Test Cases	
2019	[51]	RAZOR	B2B	Type II	Basic Block	Test Cases	•
2022	[5, 70]	BinRec-ToB	B2B	Type II	Basic Block	Test Cases	•
2023	[26]	GTIRB Binary Reduce (Dynamic)	B2B	Type II	Basic Block	Test Cases	
2021	[2, 61]	Trimmer (v2)	CBS	Type II	Instruction	Command	•
2022	[42, 45]	OCCAM (v2)	CBS	Type II	Instruction	Command	•
2023	[3]	LMCAS-SIFT	CBS	Type I/II	Instruction	Command	
2023	[26]	GTIRB Binary Reduce (Static)	SL	Type I	Function	None	
2019	[62]	Libfilter	SL	Type I	Function	None	•

Metric Selection

Table 3: Metrics Selected for Evaluation

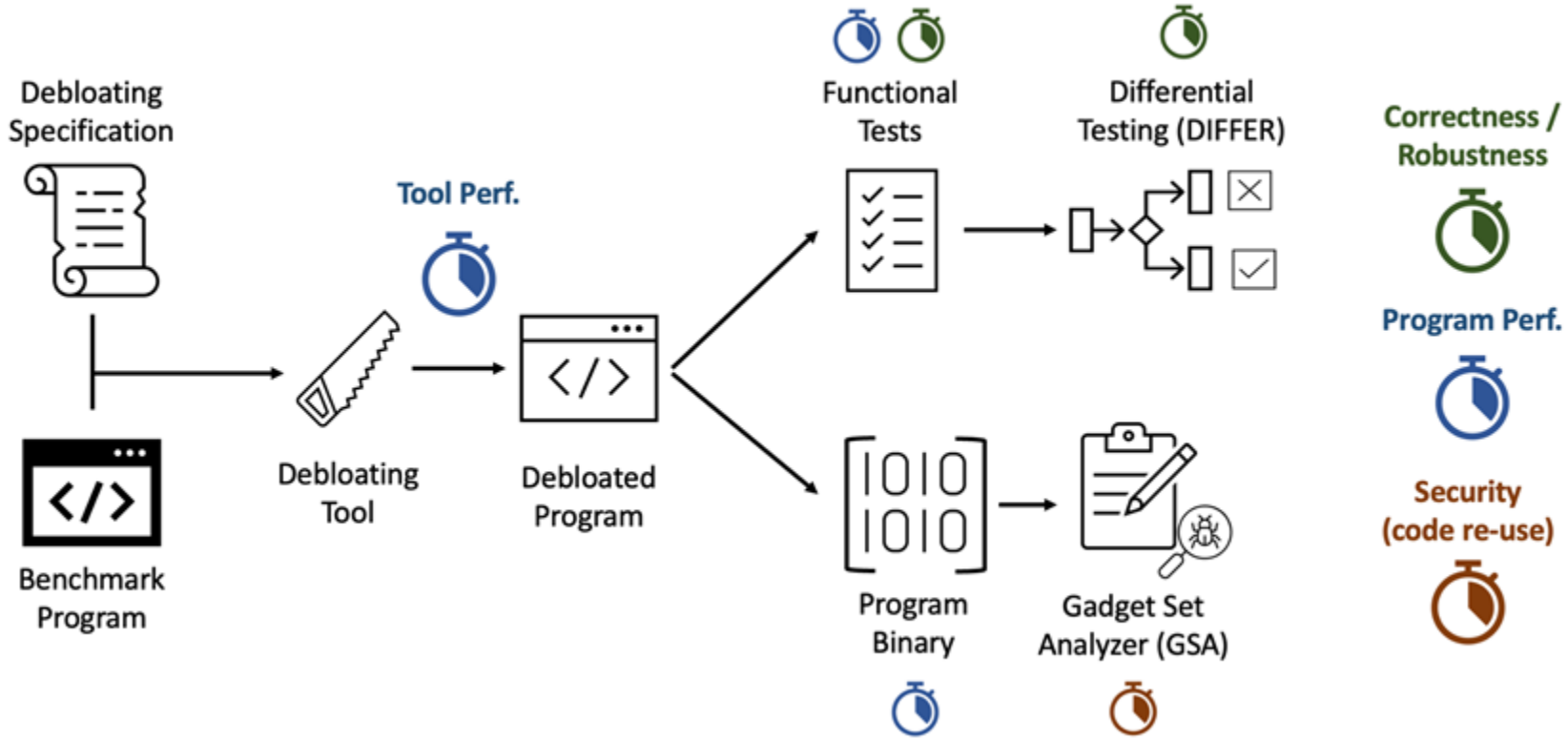
Name	Category	Name	Category
Run time (tool)	Performance	Gadget Set Expressivity	Security
Peak Memory Use (tool)	Performance	Gadget Set Quality	Security
Run time	Performance	Gadget Set Locality	Security
Peak Memory Use	Performance	Special Purpose Gadget Types Available	Security
Static Binary Size	Performance	Executes Retained Functions	Correctness / Robustness
Number of Libraries Linked	Performance	Errors / Crashes during Differential Testing	Correctness / Robustness

Benchmark Selection

Table 2: Evaluation Benchmarks

Low Complexity (CHISELBench [6])				Medium Complexity		High Complexity	
Benchmark	**Size** (KLOC)	**Benchmark**	**Size**	**Benchmark**	**Size**	**Benchmark**	**Size**
bzip2 v1.0.5	12.2	rm v8.4	7.4	bftpd v6.1	4.7	nmap v7.93	233.4
chown v8.2	7.3	sort v8.16	14.7	wget v1.20.3	14.2	nginx v1.23.3	170.6
date v8.21	9.9	tar v1.14	31.3	make v4.2	24.6	pdftohtml v0.60	16.1
grep v2.19	23.8	uniq v8.16	7.4	objdump v2.40	59.7	ImageMagick v7.0.1	361.9
gzip v1.2.4	8.9			memcached v1.6.18	30.5		
mkdir v5.2.1	5.1			lighttpd v1.4	89.7		

Evaluation Setup

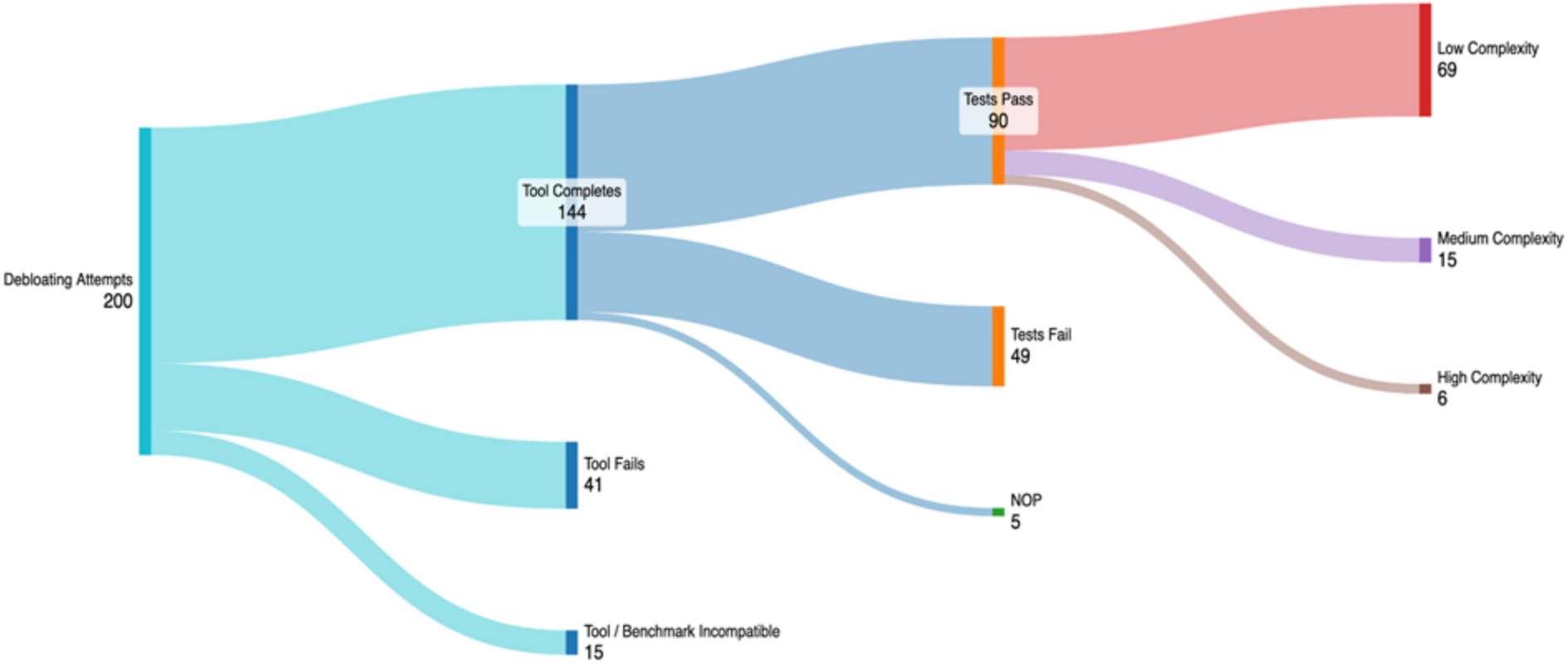


Results

How well did tools perform?

- **Only 15 tool / benchmark incompatible combinations**
  - C++, Multithreading
- **More complexity -> more resources**
- **Takes less than 20 mins and 4 GB memory to run**
  - Notable exception: CHISEL S2S debloaters take hours / days to run
  - some benchmark outliers

How well did debloated programs perform?

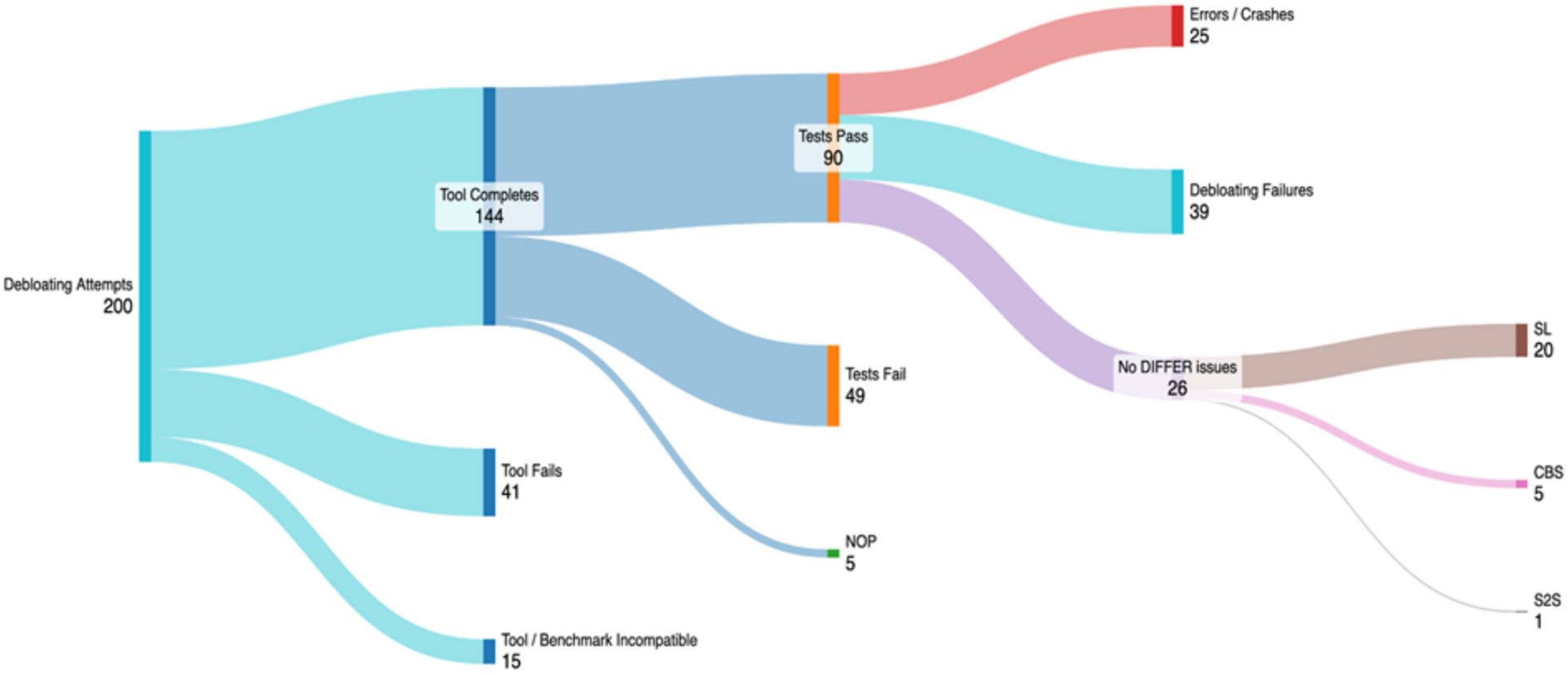


How well did debloated programs perform?

- **Reductions in static binary size as expected**
  - Come tools increase size due to design decisions

- **CPU runtime and peak memory consumption not materially changed before / after debloating**
  - As expected - the code being removed is unnecessary

How safe was debloating?



How did debloating affect security posture?

- **Debloating has mixing effect that breaks portability of code reuse exploits**

- **Other code reusability metrics were not materially impacted by debloating**

Key Findings

Key Takeaways

1. **Software debloaters currently have low maturity**
   - Slim 42.5% overall success rate passing functionality tests
   - Drops to 22% when excluding low-complexity benchmarks

2. **Software debloaters have soundness issues**
   - Only 26 of 200 attempts produced a sound debloated program
   - 20 of those were attempts to remove Type I bloat

3. **Software debloaters have marginal benefits**
   - Only binary size and gadget locality are routinely improved

Contact

Michael D. Brown

Principal Security Engineer

michael.brown@trailofbits.com