Decker: Attack Surface Reduction via On-demand Code Mapping

Chris Porter, Sharjeel Khan, Santosh Pande

Georgia Institute of Technology

ASPLOS '23, March 25–29, 2023 Vancouver, Canada

Problem

Software is bloated with unused code.

- This unneeded code contains **gadgets**.
- Gadgets are code snippets that can be stitched together to form **gadget chains** that execute malicious behavior.
- Although known CVEs are patched, future attacks can be built over these chains.

How does one remove this unwanted code and reduce the attack surface available to bad actors?

Motivation

Current **debloat** techniques remove code from either:

- libraries (achieving strong attack surface reduction), or
- applications – but at the expenses of soundness.

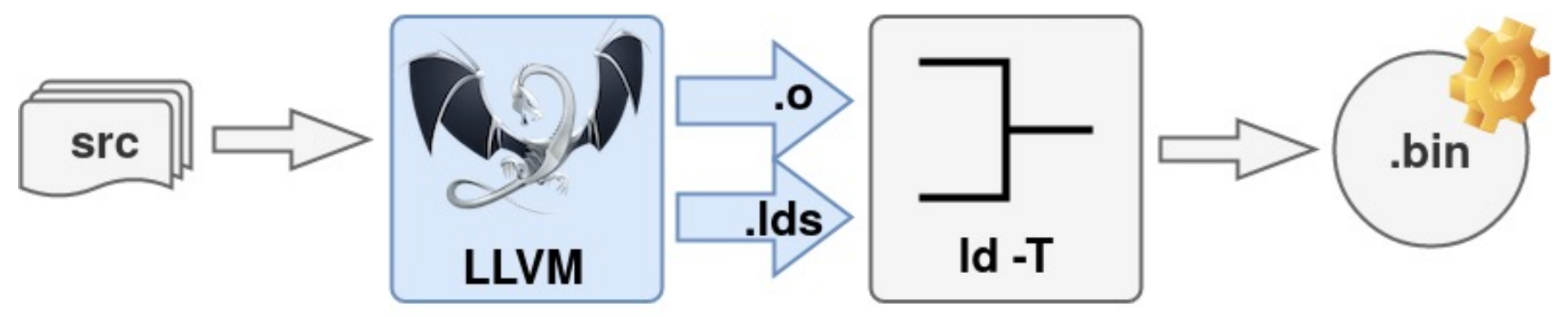
There is no general technique that:

- works on the applications as a whole instead of libraries
- is sound, and
- can effectively debloat **may-use** code using dynamic contexts.

	Piece-wise	Chisel	Razor	BlankIt
Works on application		✓	✓	
Works on library	✓		✓	✓
Works on binary			✓	✓
No user input needed	✓			✓
No training needed	✓		✓	
Is sound	✓			✓
Can debloat may-use code				✓

Overview

We propose a compiler-runtime hybrid technique.



At build time, our LLVM pass will produce a modified object file and custom linker script.

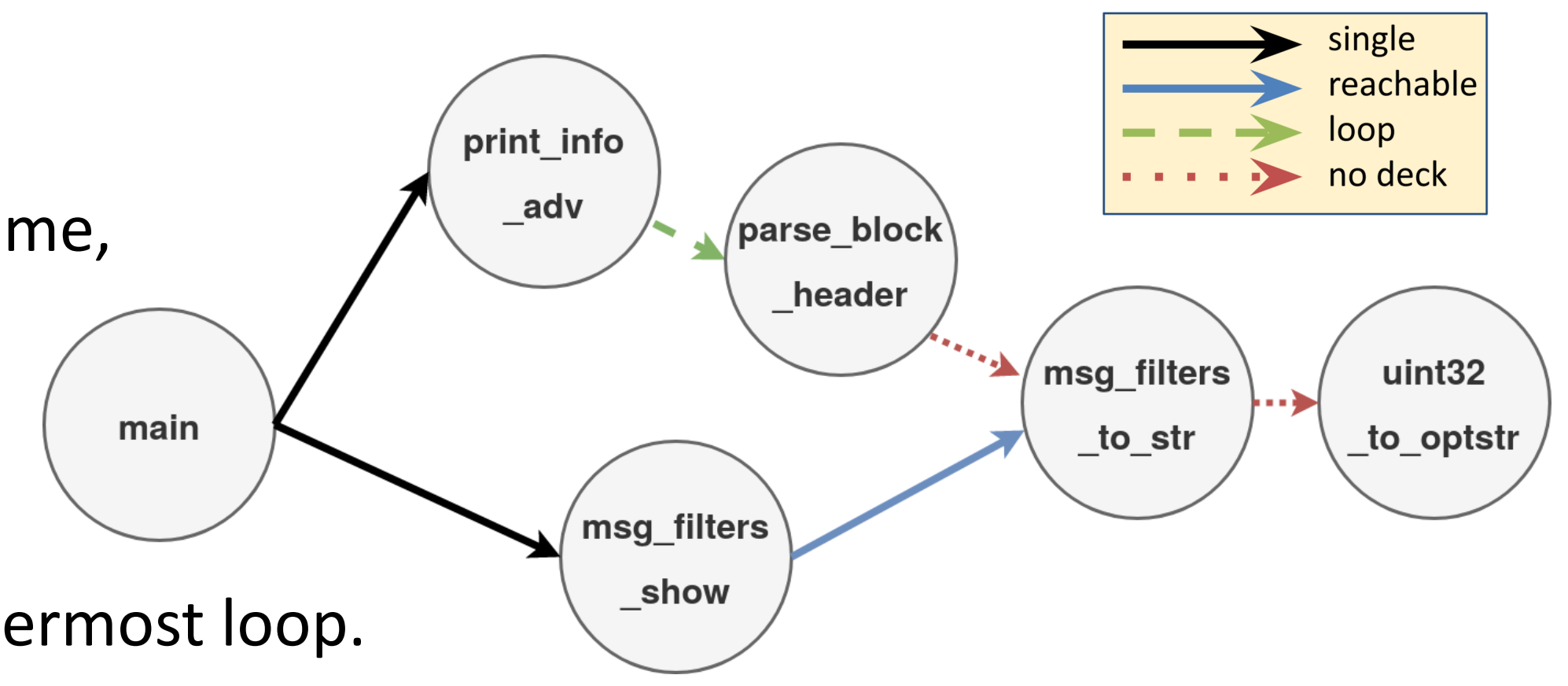
The modified program will leverage a runtime system to map upcoming regions of code that are needed (and unmap code that is unneeded).

Decker

Compiler pass

Goal: Identify regions of code that execute together. Instrument those points with function calls to the runtime, which will map those **decks** (groups of functions) as active only when needed. Define 4 types of decks:

1. Single – Only 1 function.
2. Loop – All functions within an interprocedurally outermost loop.
3. Reachable – All functions that are statically reachable by some loop.
4. Indirect – All functions that are statically reachable from some runtime-resolved indirect call.



Linking

Goal: Separate decks into **disjoint sets** so that they can be placed into separate system pages.

- Without this change, system pages could inadvertently include multiple functions that are not part of the same deck.
- Mapping system pages with unneeded functions would increase attack surface unnecessarily.
- Provide a custom linker script to the linker; the linker is unmodified.

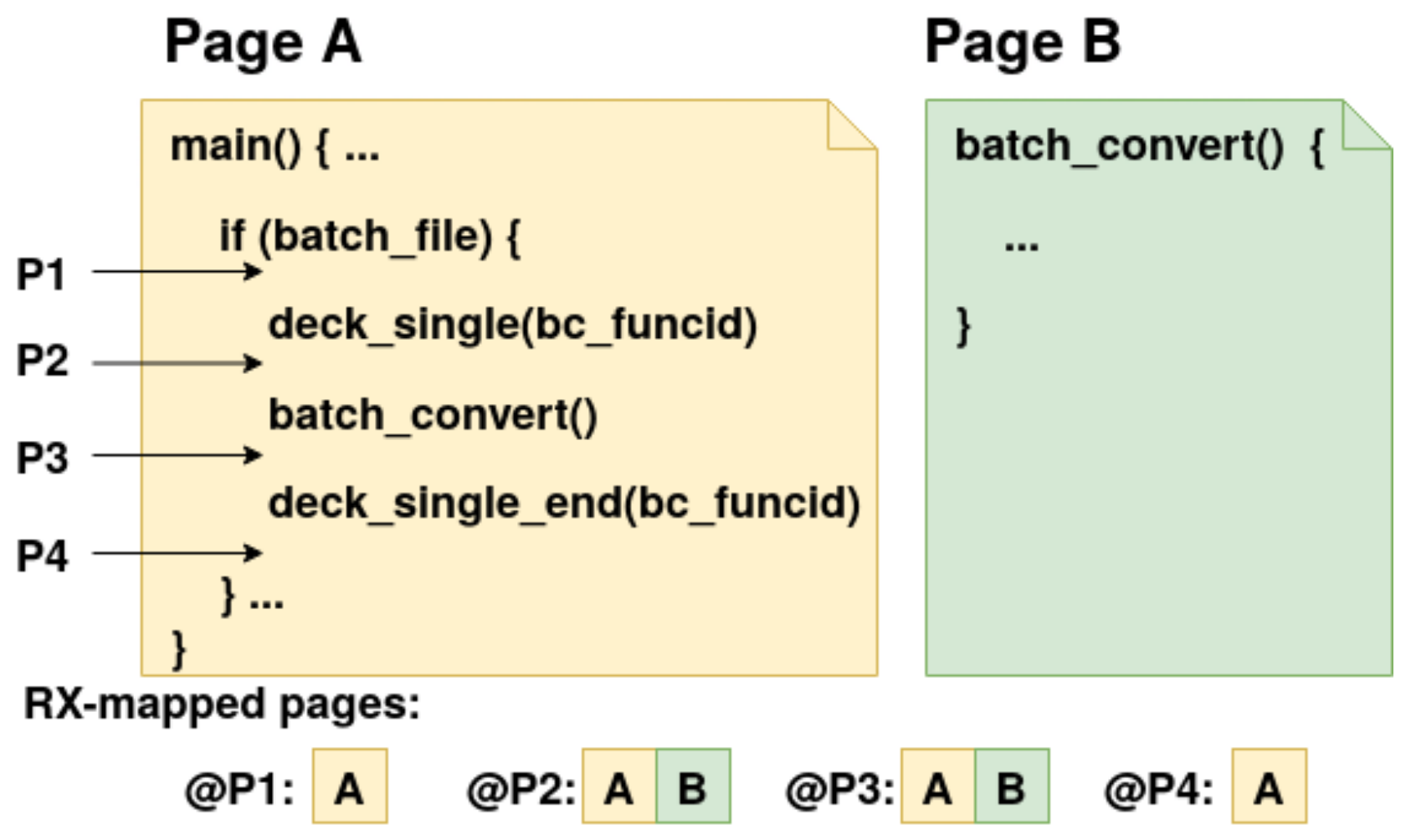
Dk.S1 = {print_info_adv}
Dk.S2 = {msg_filters_show}
Dk.L = {parse_block_header, msg_filters_to_str, uint32_to_optstr}
Dk.R = {msg_filters_to_str, uint32_to_optstr}

Dj.S1 = {print_info_adv}
Dj.S2 = {msg_filters_show}
Dj.L.1 = {parse_block_header}
Dj.I.LR = {msg_filters_to_str, uint32_to_optstr}

Runtime

Goal: Provide API and runtime support for mapping system pages of a deck as RX/RO when needed/unneeded.

- Compiler's start- and end-deck instrumentation guarantees functions are available only when needed.
- Use statically known IDs to identify all functions associated with a deck.
  - Exception: Indirect calls; use the function pointer address in this case.



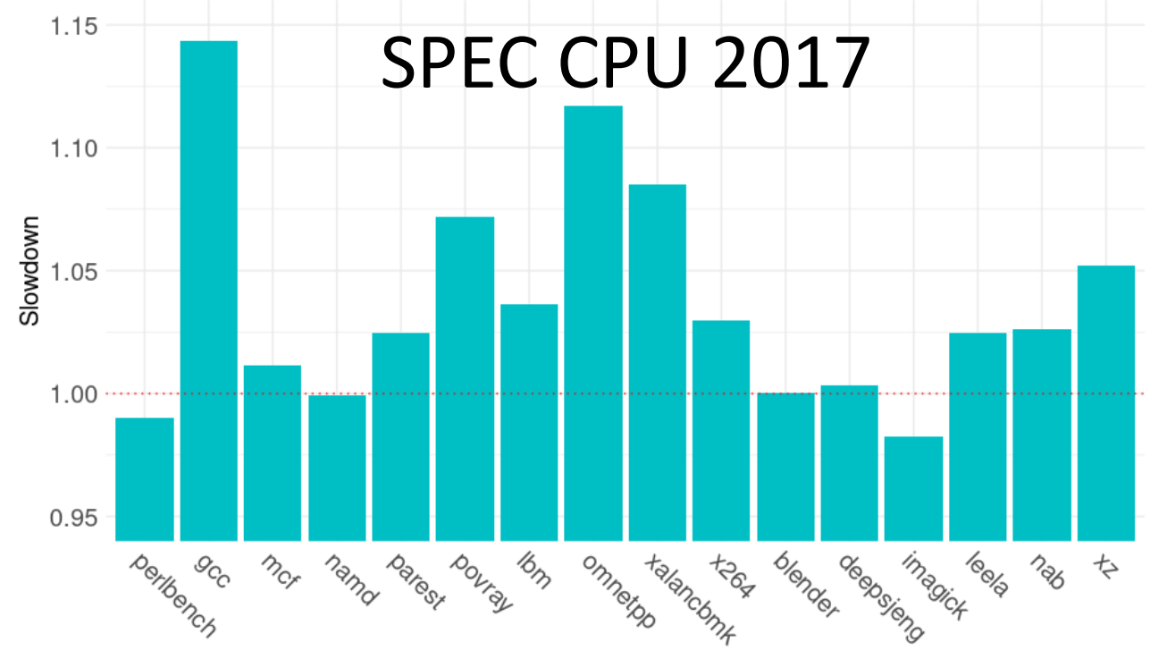
```
main() { ...
  if (batch_file) {
    deck_single(bc_funcid)
    batch_convert()
    deck_single_end(bc_funcid)
  } ...
}
```

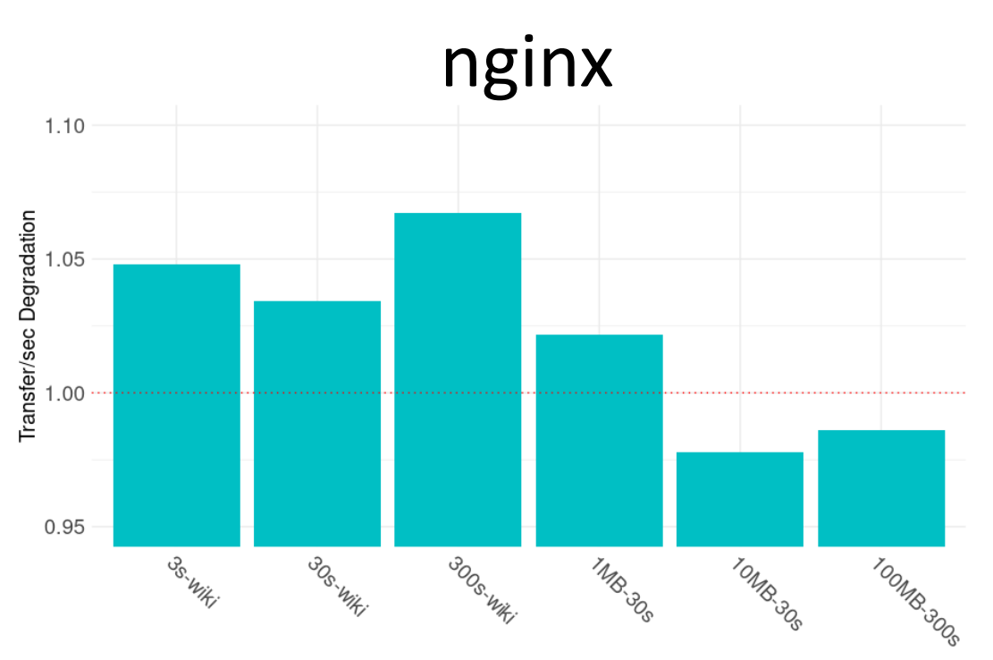
```
batch_convert() {
  ...
}
```

Results

Performance

SPEC CPU 2017: 5.2% avg overhead
GNU coreutils: negligible overhead
nginx: 2.3% avg transfer/sec degradation





Total gadget reduction

Benchmark (suite)	Min	Max	Avg
SPEC CPU 2017	49.9	91.3	73.2
GNU coreutils	72.5	91.0	87.2
nginx	50.3	95.3	80.3

Breaking gadget chains

ROP:

- 9 benchmarks, including nginx, contain shell-spawning chain.
- We analyze every available page set over all inputs across all applications with Ropper and find that Decker does not allow the ROP chain under any set of dynamic decks.
- Analysis includes 6,453 unique dynamic deck sets (with 14,378 dynamic execution count).

JOP

- Both Linux and Windows studies on nginx.
- Decker consistently restricts useful gadgets from being used together (e.g. mov cannot be used with pop gadgets).

Conclusion

- Decker is a **sound** attack surface reduction technique for applications that requires no training, user inputs, or specifications.
- It achieves strong total gadget reductions (>70%) at low overhead (~5% or less) across SPEC, coreutils, and nginx.
- It breaks ROP shell spawning chains in all experiments, and there is strong evidence that useful JOP chains are substantially hampered or impossible to build for both Linux and Windows when using Decker.