A Multi-OS Cross-Layer Study of Bloating in User Programs, Kernel and Managed Execution Environments

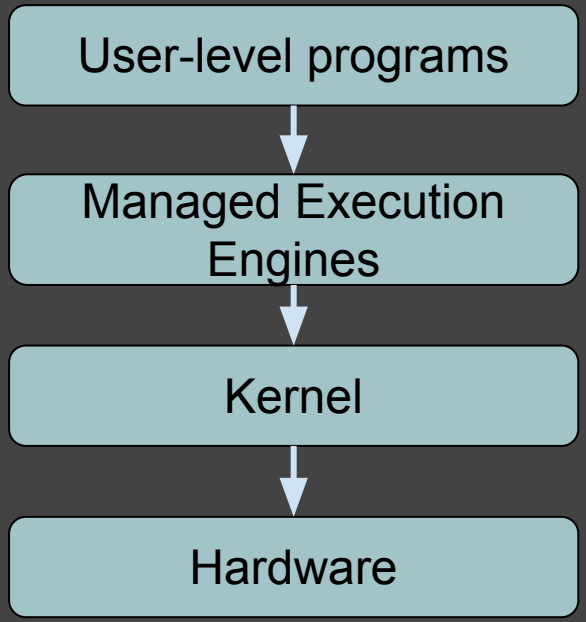
Anh Quach, Rukayat Erinfolami,

David Demicco, Aravind Prakash

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Code Bloating in Software Stack

- Bloating: unused resource in memory
  - Management cost
  - Vulnerabilities
  - Gadget source
- Modular and abstraction
- Occurs across layers



Bloat in User-level Programs

- Bad coding practice --- redundant copies of code, unnecessary inlining, etc.
- Feature-rich generic libraries (e.g., libc, BOOST, FFMpeg)
- Static Approach --- What is the minimum amount of bloat in the program?
  - Recursive dependency analysis
  - Include address-taken functions for indirect code references
- Dynamic Approach --- How much code executes for an average workload?
  - Dynamic analysis in a controlled execution monitor (Pin for user programs and QEMU for kernel)

Lower Bound Bloat Results (Using Static Analysis)

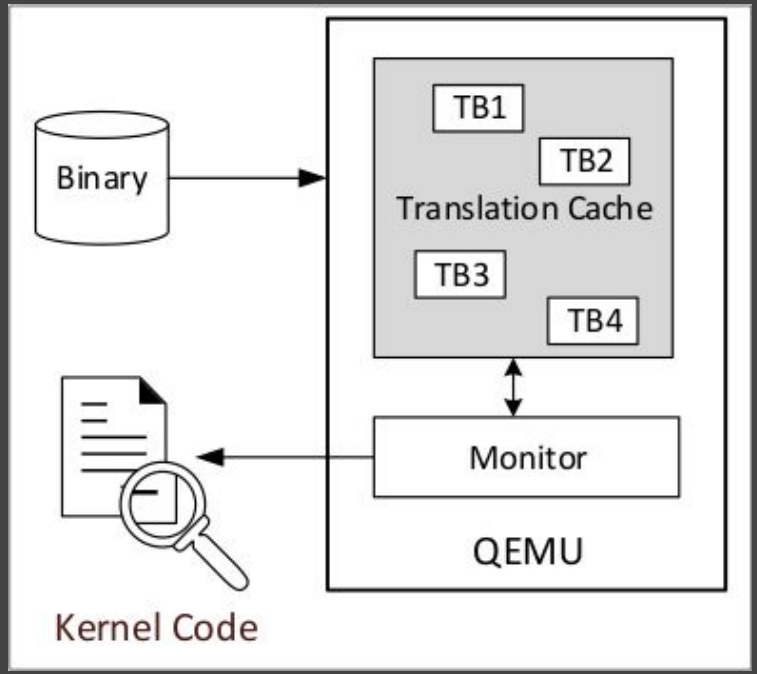
Program	% Library Instructions Required	% Overall Instructions Required	% Library Functions Required	% Overall Functions Required
firefox	67.20%	68.37%	36.42%	38.60%
chrome	69.72%	95.67%	33.57%	36.75%
webbrowser-app	58.86%	59.03%	29.34%	30.22%
vlc	78.22%	78.25%	42.44%	42.79%
rhythmbox	77.92%	77.92%	29.83%	29.83%
evince	70.84%	71.34%	33.61%	36.19%
sublime	68.88%	84.95%	39.13%	41.42%
gnome-calculator	68.59%	69.21%	34.02%	36.18%
git	62.70%	78.11%	22.75%	29.11%
clang	53.99%	73.91%	34.32%	56.83%
g++	52.36%	64.37%	23.90%	29.58%
make	52.13%	56.06%	23.11%	27.75%
Average	**65.12%**	**73.10%**	**31.87%**	**36.27%**

User-level Programs Average Bloat Results

Program	Workload	% Instructions Executed in Libraries	% Overall Instructions Executed	% Functions Executed in Libraries	# Unique Syscalls (out of 402)
firefox	Open top 10 websites in Alexa's list	28.66%	28.70%	17.04%	101
webbrowser-app	Open google.com and youtube.com. Play a video on youtube.com.	12.70%	12.76%	15.28%	93
vlc	Play 1 song	12.44%	12.44%	10.54%	80
libreoffice	Create, write and save a new word file.	23.41%	23.41%	16.03%	86
sublime	Create, write and save a new word file.	26.66%	38.12%	16.85%	67
gnome-calculator	Add and subtract numbers.	35.18%	36.25%	21.35%	59
git	Clone a respository	12.61%	11.78%	6.71%	47
clang++	Compile a C++ program	6.63%	10.32%	8.62%	23
g++	Compile a C++ program	4.52%	17.57%	2.53%	17
make	Run make on a C++ project.	11.97%	18.20%	6.22%	26
Average		**17.48%**	**20.96%**	**12.12%**	**59.9**

Measuring Bloat in Kernel

- Linux kernel: “bloated and huge”*, monolithic
- Measuring kernel bloat is hard! [DECAF, TSE’16]
- Light-weight Qemu-based kernel tracer
  - Intercept translation state
  - No need to acquire CPU state



* Torvalds, Linus. LinuxCon, 2009

Bloating in Kernel Results

System Call	Kernel Code Executed (B)		% Kernel Code Executed	
	kFreeBSD	Debian	kFreeBSD	Debian
exit	941961	778361	8.92%	7.89%
exit_group	Not Supported	462053	Not Supported	4.68%
open+close	1076732	964614	10.20%	9.78%
getuid	792612	575879	7.51%	5.84%
execve	1453331	1388650	13.77%	14.07%
getcwd	681763	903860	6.46%	9.16%
write	792519	713358	7.51%	7.23%
getpid	670177	533857	6.35%	5.41%
Average	**915585**	**836939.9**	**8.67%**	**8.48%**

Bloating in Managed Execution Engines

- Designed to support all features.
- Managed programs may use some features
- Dynamic approach: code executed for a typical workload

Bloating in JVM

Program	Workload	# Modules in JVM	% Instructions Executed	% Functions Executed
Eclipse	Created a java program, compiled and executed it.	6	10.50%	25.13%
		8	33.65%	35.45%
Jabref	Created a bibliography file	13	31.88%	30.42%
Jenkins	Installed plugins, created an admin user, and created a pipeline	12	34.51%	32.39%
Average		**10**	**27.63%**	**30.85%**

Bloating in Python Interpreter

Program	Workload	# Modules in python	% Instructions Executed	% Functions Executed
Calibre	Opened and read a book	34	6.75%	4.97%
		34	7.08%	7.66%
Mercurial	Cloned a repository	4	39.76%	69.23%
		6	40.84%	44.44%
Pip	Installed a program	11	14.52%	60.49%
		11	9.21%	60.49%
Ubuntu software center	Installed and removed a program	2	11.33%	53.85%
		16	32.75%	14.32%
		17	44.88%	30.07%
Gramps	Created a family tree	8	54.25%	36.67%
Average		**14.3**	**26.14%**	**38.22%**

Debloating Approaches and Challenges

- Partition problem space into Binary and Source code
  - Wide use of opensource libraries.
- Partition shared libraries into smaller libraries
- Challenges for code removal:
  - Semantic gap across layers, Code sharing, Context sensitive approach
- Quantifying the security benefits of debloating

Questions