Is Less *Really* More?
Towards Better Metrics for Measuring Security Improvements Realized Through Software Debloating

Michael D. Brown
Georgia Tech Research Institute (GTRI)

Professor Santosh Pande
Georgia Institute of Technology

At a High Level

- Software Debloating is an emerging cyber hardening technique that removes unnecessary parts of a program to reduce its attack surface.
- A frequently cited security improvement metric used in recent work is code reuse gadget count reduction.
- This metric is *quantitative* in nature and can be misleading; Software debloating introduces new gadgets and can actually make gadget sets more useful.
- We propose two *qualitative* metrics for measuring gadget set properties and present our tool for measuring the change in these properties that results from debloating.
- Our data indicates that debloating to improve security isn't as straightforward as once thought.

What is Software Bloat?

Modern software engineering practices favor software and systems that are:

- Modular
- Reusable
- Feature Rich

This helps engineers rapidly develop complex and widely deployable software.

However, it comes at a cost – when software is deployed and executed it contains large portions of code that will never be used. This is software bloat.

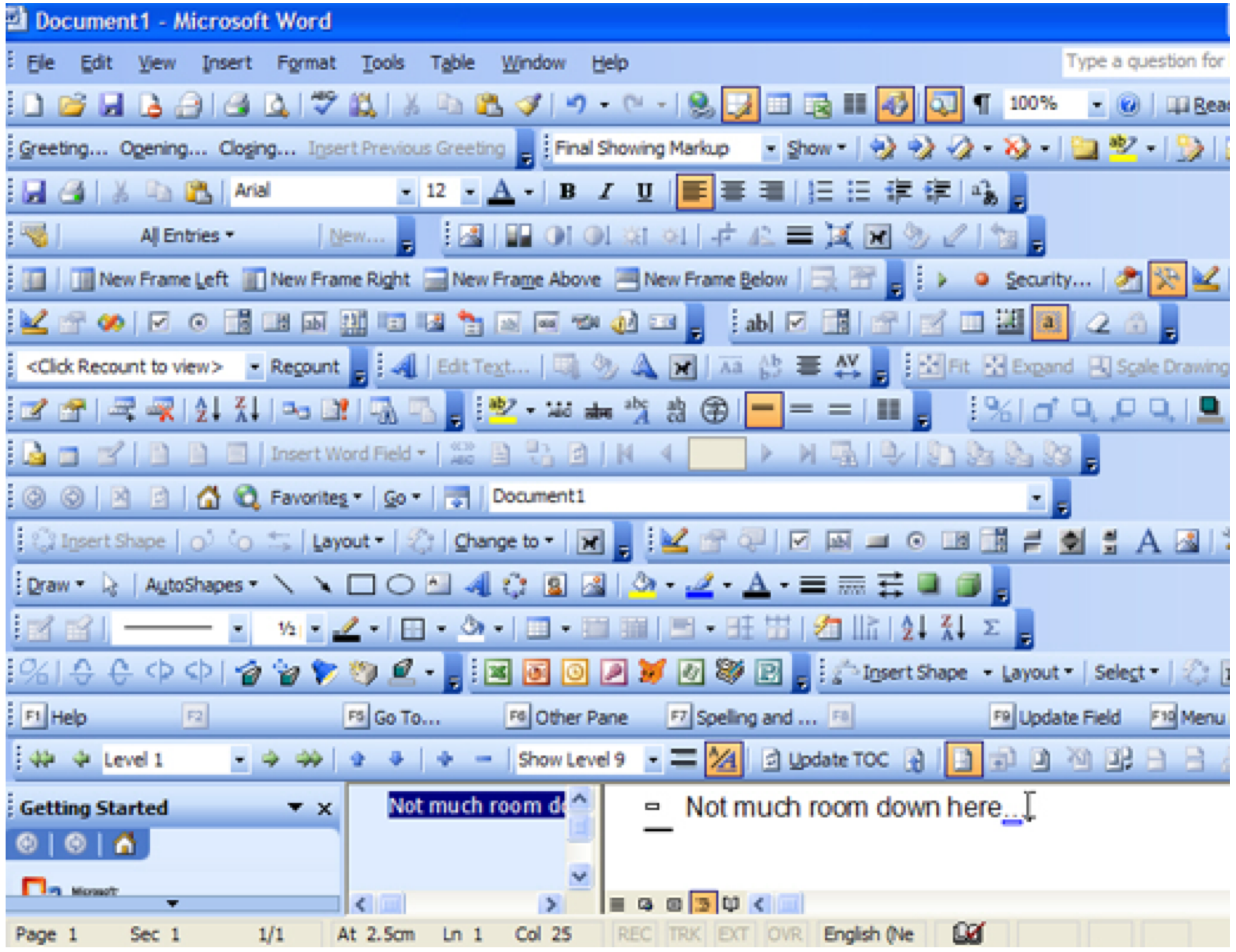
Sources of Software Bloat: Feature Creep

Software bloat occurs laterally within a software package due to feature creep.

Example: `cUrl` supports data transfer via 23 different protocols:

DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, Telnet and TFTP

Common users of `cUrl` or `libcurl` are unlikely to full utilize this level of feature variety.


Document1 - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Type a question for
Not much room down here...

Prevalence and Impact of Software Bloat

A recent study of vertical software bloat by Quach et al [1] found:

Commonly used features in these programs require a relatively small portion of the total instructions present in the programs and libraries. For example:

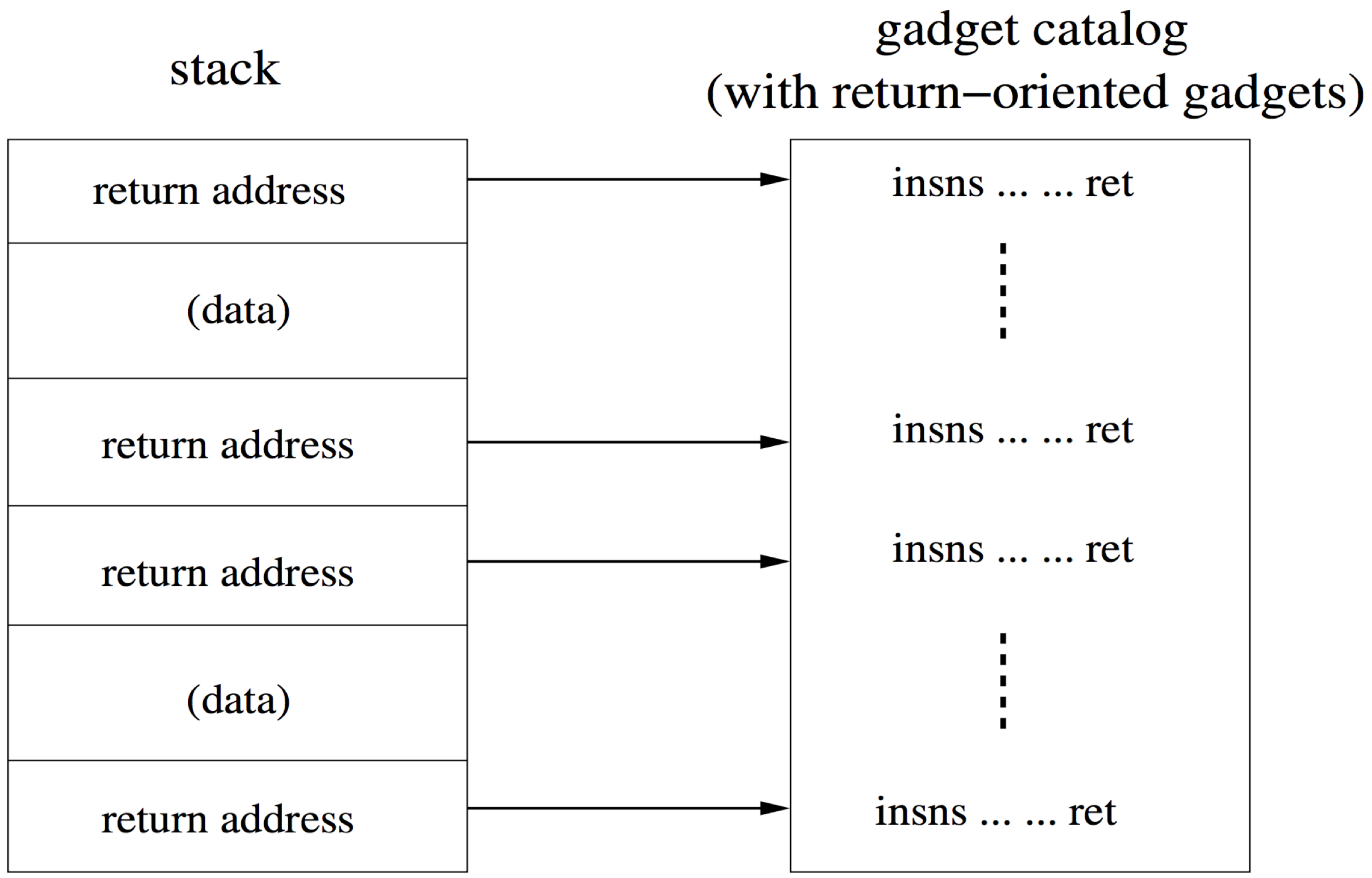
- Playing an audio file in VLC requires only 12% of the overall instructions.
- Creating, composing, and saving a file in Sublime requires only 27% of the overall instructions.
- Fetching and displaying 10 popular websites in Firefox requires only 29% of the overall instructions.

Bloat has a number of negative security impacts, namely:

- Bloat code may contain reachable attack vectors / vulnerabilities
- Bloat code may increase the overhead of security defenses
- **Bloat code can potentially be used in a code reuse attack**

Software Bloat and Gadgets

- Gadgets are used like complex instructions, and the the total set of gadgets available to the attacker is their ISA.
- Gadgets end in a return, indirect call, or indirect jump instruction, which the attacker can exploit to maintain control flow.
- This bloat code is useless to the user executing the program, but contributes to the total set of gadgets available to the attacker.
- The attacker can use these gadgets in the construction of their exploits.

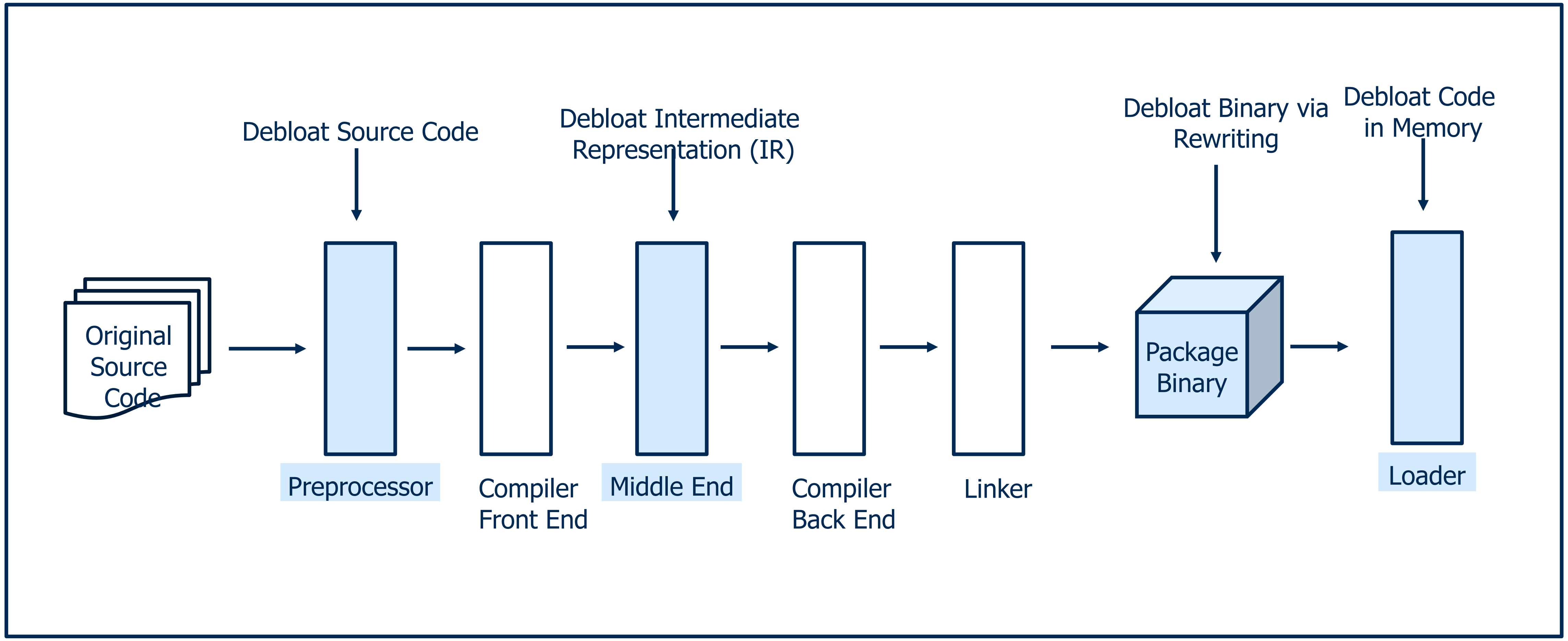


(a) The ROP model

Software Debloating

Software debloating is a software transformation that produces variant of a program that contains the minimum amount of code necessary for its specific end use context.

Software debloating can be performed at many different points in the software lifecycle:



Difficulties in Measuring Claims of Improved Security

- Very difficult to measure security improvement with respect to vulnerability elimination and moving target defense.

- Measuring code reuse gadget count reduction is easy, and makes sense on the surface
  - Less code means fewer pieces of code the attacker can stitch together in gadget based code reuse attacks

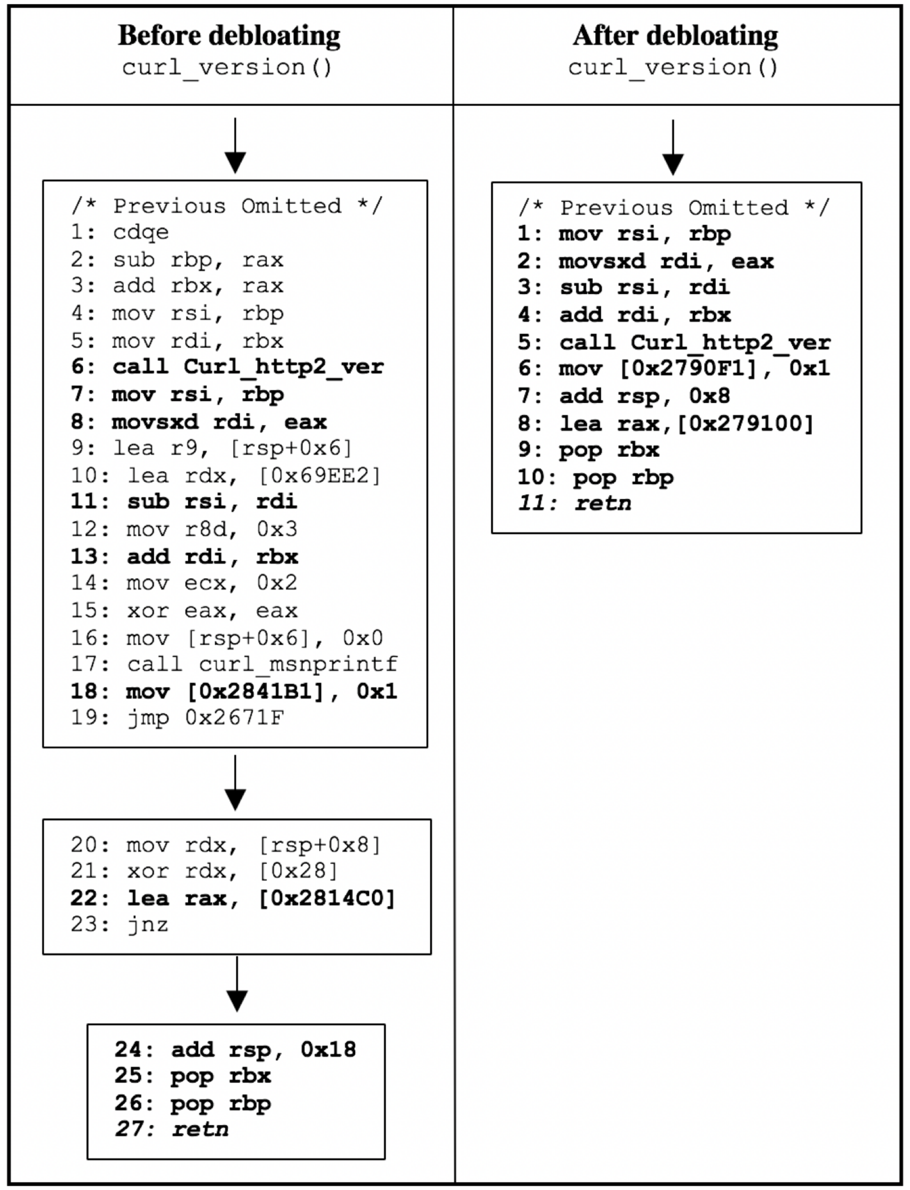
- However:
  - Reducing number of gadgets doesn’t necessarily make creating an exploit harder
  - Attackers don’t need large, diverse sets of gadgets to craft exploits
  - Code-removing debloaters can *introduce new gadgets!*

Gadget Introduction Mechanisms – Compiler Optimization

- Removing code from a software package via debloating can have unpredictable effects on optimization and code generation choices made by the compiler.

- Some optimizations suppressed and/or triggered by debloating:
  - Loop Unrolling
  - Function Inlining
  - Dead Code Elimination

```
char *curl_version(void)
{
  static bool initialized;
  static char version[200];
  char *ptr = version;
  size_t len;
  size_t left = sizeof(version);
  if(initialized)
    return version;
  strcpy(ptr,
         LIBCURL_NAME "/" LIBCURL_VERSION);
  len = strlen(ptr);
  left -= len;
  ptr += len;
  /* Similar version checks omitted */
  len = Curl_http2_ver(ptr, left);
  left -= len;
  ptr += len;
  /* Start of segment to be debloated */
  char suff[2];
  if(RTMP_LIB_VERSION & 0xff) {
    suff[0]=(RTMP_LIB_VERSION & 0xff)
             + 'a' - 1;
    suff[1]= '\0';
  }
  else
    suff[0] = '\0';
  snprintf(ptr, left, " librtmp/%d.%d%s",
           RTMP_LIB_VERSION >> 16,
           (RTMP_LIB_VERSION >> 8)
           & 0xff, suff);
  /* End of segment to be debloated */
  initialized = true;
  return version;
}
```



Before debloating
curl_version()

```
/* Previous Omitted */
cdqe
sub rbp, rax
add rbx, rax
mov rsi, rbp
mov rdi, rbx
call Curl_http2_ver
mov rsi, rbp
movsxd rdi, eax
lea r9, [rsp+0x6]
lea rdx, [0x69EE2]
sub rsi, rdi
mov r8d, 0x3
add rdi, rbx
mov ecx, 0x2
xor eax, eax
mov [rsp+0x6], 0x0
call curl_msnprintf
mov [0x2841B1], 0x1
jmp 0x2671F
```

```
mov rdx, [rsp+0x8]
xor rdx, [0x28]
lea rax, [0x2814C0]
jnz
```

```
add rsp, 0x18
pop rbx
pop rbp
retn
```

After debloating
curl_version()

```
/* Previous Omitted */
mov rsi, rbp
movsxd rdi, eax
sub rsi, rdi
add rdi, rbx
call Curl_http2_ver
mov [0x2790F1], 0x1
add rsp, 0x8
lea rax,[0x279100]
pop rbx
pop rbp
retn
```

Gadget Introduction Mechanisms – Unintended Gadgets

- x86 and x86-64 have variable length instructions, so it is possible to decode instructions from an offset other than the original instruction boundary. Gadgets found using this method are called unintended gadgets.
- Since debloating causes significant changes to the package's final representation, the unintended gadgets in a package can vary greatly in a debloated variant.

```
48 83 c4 18:  add rsp,0x18
5b:           pop rbx
5d:           pop rbp
c3:           ret
```

```
83 c4 18:     add esp,0x18
5b:           pop rbx
5d:           pop rbp
c3:           ret
```

```
18 5b 5d:     sbb BYTE PTR[rbx+0x5d],bl
c3:           ret
```

```
c6 05 0f 4d
24 00 01:          mov BYTE PTR[rip+0x244d0f],0x1
48 83 c4 08:       add rsp,0x8
48 8d 05 13
4d 24 00:          lea rax,[rip+0x244d13]
5b:                pop rbx
5d:                pop rbp
c3:                ret
```

```
08 48 8d :         or BYTE PTR [rax-0x73],cl
05 13 4d 24 00:    add eax,0x244d13
5b:                pop rbx
5d:                pop rbp
c3:                ret
```

```
13 4d 24:          adc ecx,DWORD PTR [rbp+0x24]
00 5b 5d:          add BYTE PTR[rbx+0x5d],bl
c3:                ret
```

How Prevalent is Gadget Introduction?

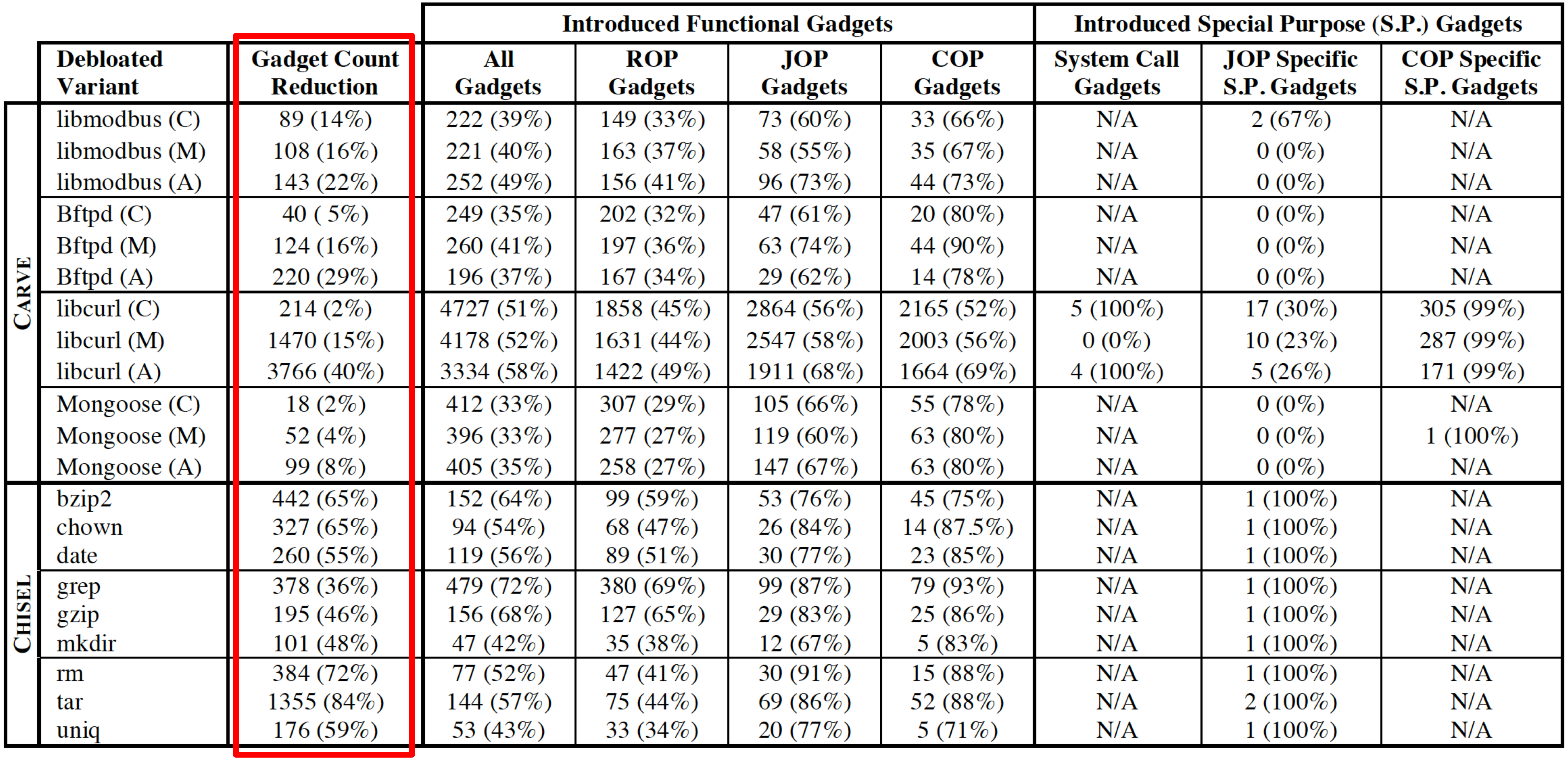
- We debloated thirteen software packages that varied in size, structure and operational complexity using two different code-removing debloaters.
  - CHISEL (Linux CoreUtils)
  - CARVE (Network Protocol Implementations)

- CARVE benchmark packages were debloated at three different intensity levels:
  - Conservative: Some peripheral features in the package are targeted for debloating.
  - Moderate: Some peripheral features and some core features are targeted for debloating.
  - Aggressive: All debloatable features except for a small set of core features are targeted for debloating.

- CHISEL benchmarks debloated at level roughly equivalent to Aggressive.

- Both debloaters achieve comparable gadget count reduction rates.

Prevalence of Gadget Introduction



			Introduced Functional Gadgets				Introduced Special Purpose (S.P.) Gadgets		
	Debloated Variant	Gadget Count Reduction	All Gadgets	ROP Gadgets	JOP Gadgets	COP Gadgets	System Call Gadgets	JOP Specific S.P. Gadgets	COP Specific S.P. Gadgets
CARVE	libmodbus (C)	89 (14%)	222 (39%)	149 (33%)	73 (60%)	33 (66%)	N/A	2 (67%)	N/A
	libmodbus (M)	108 (16%)	221 (40%)	163 (37%)	58 (55%)	35 (67%)	N/A	0 (0%)	N/A
	libmodbus (A)	143 (22%)	252 (49%)	156 (41%)	96 (73%)	44 (73%)	N/A	0 (0%)	N/A
	Bftpd (C)	40 (5%)	249 (35%)	202 (32%)	47 (61%)	20 (80%)	N/A	0 (0%)	N/A
	Bftpd (M)	124 (16%)	260 (41%)	197 (36%)	63 (74%)	44 (90%)	N/A	0 (0%)	N/A
	Bftpd (A)	220 (29%)	196 (37%)	167 (34%)	29 (62%)	14 (78%)	N/A	0 (0%)	N/A
	libcurl (C)	214 (2%)	4727 (51%)	1858 (45%)	2864 (56%)	2165 (52%)	5 (100%)	17 (30%)	305 (99%)
	libcurl (M)	1470 (15%)	4178 (52%)	1631 (44%)	2547 (58%)	2003 (56%)	0 (0%)	10 (23%)	287 (99%)
	libcurl (A)	3766 (40%)	3334 (58%)	1422 (49%)	1911 (68%)	1664 (69%)	4 (100%)	5 (26%)	171 (99%)
	Mongoose (C)	18 (2%)	412 (33%)	307 (29%)	105 (66%)	55 (78%)	N/A	0 (0%)	N/A
	Mongoose (M)	52 (4%)	396 (33%)	277 (27%)	119 (60%)	63 (80%)	N/A	0 (0%)	1 (100%)
	Mongoose (A)	99 (8%)	405 (35%)	258 (27%)	147 (67%)	63 (80%)	N/A	0 (0%)	N/A
CHISEL	bzip2	442 (65%)	152 (64%)	99 (59%)	53 (76%)	45 (75%)	N/A	1 (100%)	N/A
	chown	327 (65%)	94 (54%)	68 (47%)	26 (84%)	14 (87.5%)	N/A	1 (100%)	N/A
	date	260 (55%)	119 (56%)	89 (51%)	30 (77%)	23 (85%)	N/A	1 (100%)	N/A
	grep	378 (36%)	479 (72%)	380 (69%)	99 (87%)	79 (93%)	N/A	1 (100%)	N/A
	gzip	195 (46%)	156 (68%)	127 (65%)	29 (83%)	25 (86%)	N/A	1 (100%)	N/A
	mkdir	101 (48%)	47 (42%)	35 (38%)	12 (67%)	5 (83%)	N/A	1 (100%)	N/A
	rm	384 (72%)	77 (52%)	47 (41%)	30 (91%)	15 (88%)	N/A	1 (100%)	N/A
	tar	1355 (84%)	144 (57%)	75 (44%)	69 (86%)	52 (88%)	N/A	2 (100%)	N/A
	uniq	176 (59%)	53 (43%)	33 (34%)	20 (77%)	5 (71%)	N/A	1 (100%)	N/A

We Need Better Gadget Metrics for Software Debloating

Gadget count reduction is too superficial to be an accurate metric for measuring security improvement. Given the prevalence of gadget introduction, gadget count reduction is potentially misleading.

We propose qualitative (vs. quantitative) metrics that address the key question:
How does debloating make constructing a code reuse attack more challenging or difficult?

Functional Gadget Set Expressivity

- Measure change in computational power of gadget ”Instruction Set” after debloating.
- Tells us what kinds of computations can be specified by a set of gadgets.

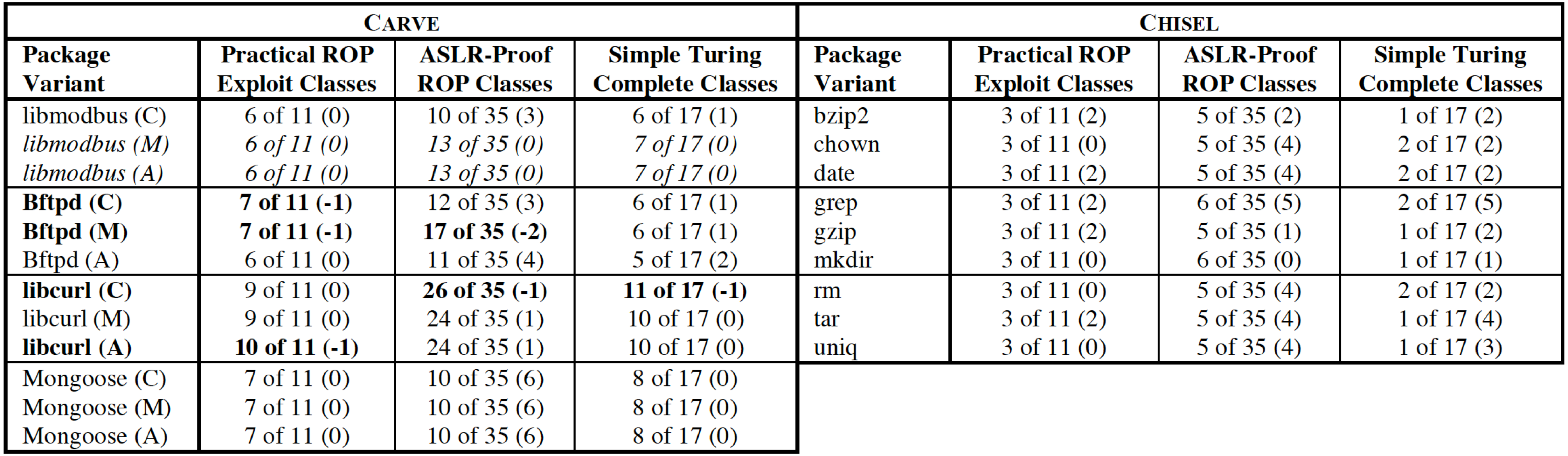
Special Purpose Gadget Availability

- Determine if debloating removes critical gadgets necessary to construct exploits.
- Tells us if the necessary infrastructure gadgets for different exploit types are present

GSA – Gadget Set Analyzer

- We created a static binary analysis tool capable of analyzing a software package and its debloated variants to capture changes in our metrics.
- GSA is built on top of existing tools: ROPgadget [6] and a gadget classifier [9].
- Performs a secondary search of discovered gadgets to identify special purpose gadgets.
- Measures the expressivity of discovered gadgets against three bars:
    - Turing Completeness
    - Practical ROP exploits
    - ASLR-proof practical ROP exploits
- We used our tool to analyze our benchmarks to get a better idea of how debloating impacted gadget set quality.

Results – Functional Gadget Set Expressivity



CARVE				CHISEL			
Package Variant	**Practical ROP Exploit Classes**	**ASLR-Proof ROP Classes**	**Simple Turing Complete Classes**	**Package Variant**	**Practical ROP Exploit Classes**	**ASLR-Proof ROP Classes**	**Simple Turing Complete Classes**
libmodbus (C)	6 of 11 (0)	10 of 35 (3)	6 of 17 (1)	bzip2	3 of 11 (2)	5 of 35 (2)	1 of 17 (2)
libmodbus (M)	*6 of 11 (0)*	*13 of 35 (0)*	*7 of 17 (0)*	chown	3 of 11 (0)	5 of 35 (4)	2 of 17 (2)
libmodbus (A)	*6 of 11 (0)*	*13 of 35 (0)*	*7 of 17 (0)*	date	3 of 11 (2)	5 of 35 (4)	2 of 17 (2)
Bftpd (C)	**7 of 11 (-1)**	12 of 35 (3)	6 of 17 (1)	grep	3 of 11 (2)	6 of 35 (5)	2 of 17 (5)
Bftpd (M)	**7 of 11 (-1)**	**17 of 35 (-2)**	6 of 17 (1)	gzip	3 of 11 (2)	5 of 35 (1)	1 of 17 (2)
Bftpd (A)	6 of 11 (0)	11 of 35 (4)	5 of 17 (2)	mkdir	3 of 11 (0)	6 of 35 (0)	1 of 17 (1)
libcurl (C)	9 of 11 (0)	**26 of 35 (-1)**	**11 of 17 (-1)**	rm	3 of 11 (0)	5 of 35 (4)	2 of 17 (2)
libcurl (M)	9 of 11 (0)	24 of 35 (1)	10 of 17 (0)	tar	3 of 11 (2)	5 of 35 (4)	1 of 17 (4)
libcurl (A)	**10 of 11 (-1)**	24 of 35 (1)	10 of 17 (0)	uniq	3 of 11 (0)	5 of 35 (4)	1 of 17 (3)
Mongoose (C)	7 of 11 (0)	10 of 35 (6)	8 of 17 (0)				
Mongoose (M)	7 of 11 (0)	10 of 35 (6)	8 of 17 (0)				
Mongoose (A)	7 of 11 (0)	10 of 35 (6)	8 of 17 (0)				

Results – Special Purpose Gadget Availability

	Package Variant	Syscall Gadget	JOP Dispatcher Gadgets	JOP Data Loader Gadgets	JOP Trampoline Gadgets	COP Dispatcher Gadgets	COP Intra Stack Pivot Gadgets
CARVE	***libmodbus (C)***	*0 (0)*	*0 (0)*	***3 (-2)***	*0 (0)*	*0 (0)*	*0 (0)*
	libmodbus (M)	*0 (0)*	*0 (0)*	*1 (0)*	*0 (0)*	*0 (0)*	*0 (0)*
	libmodbus (A)	*0 (0)*	*0 (0)*	*1 (0)*	*0 (0)*	*0 (0)*	*0 (0)*
	Bftpd (C)	0 (0)	0 (0)	9 (1)	0 (0)	0 (0)	0 (0)
	Bftpd (M)	0 (0)	0 (0)	1 (9)	0 (0)	0 (0)	0 (0)
	Bftpd (A)	0 (0)	0 (0)	1 (9)	0 (0)	0 (0)	0 (0)
	libcurl (C)	**5 (-1)**	7 (1)	50 (1)	0 (1)	304 (15)	4 (0)
	libcurl (M)	0 (4)	4 (4)	41 (10)	0 (1)	287 (32)	3 (1)
	libcurl (A)	4 (0)	3 (5)	14 (37)	1 (0)	170 (149)	2 (2)
	Mongoose (C)	*0 (0)*	***1 (-1)***	*6 (0)*	*0 (0)*	*0 (0)*	*0 (0)*
	Mongoose (M)	*0 (0)*	*0 (0)*	*6 (0)*	*0 (0)*	*0 (0)*	*0 (0)*
	Mongoose (A)	*0 (0)*	*0 (0)*	*6 (0)*	*0 (0)*	*0 (0)*	*0 (0)*
CHISEL	*bzip2*	*0 (0)*	*0 (0)*	*1 (0)*	*0 (0)*	*0 (0)*	*0 (0)*
	chown	0 (4)	0 (0)	1 (0)	0 (0)	0 (0)	0 (0)
	date	0 (4)	0 (0)	1 (0)	0 (0)	0 (0)	0 (0)
	grep	0 (4)	0 (0)	1 (3)	0 (0)	0 (0)	0 (0)
	gzip	*0 (0)*	*0 (0)*	*1 (0)*	*0 (0)*	*0 (0)*	*0 (0)*
	mkdir	0 (0)	0 (0)	1 (0)	0 (0)	0 (0)	0 (0)
	rm	0 (4)	0 (0)	1 (0)	0 (0)	0 (0)	0 (0)
	tar	0 (2)	0 (0)	**2 (-1)**	0 (0)	0 (0)	0 (0)
	uniq	0 (4)	0 (0)	1 (0)	0 (0)	0 (0)	0 (0)

Summary of Results

Analysis of the 21 debloating scenarios using our proposed metrics indicates a very different picture than gadget count reduction.

- In 5 of 21 scenarios, negative impacts were observed after debloating:
  - New special purpose gadgets that were previously unavailable are introduced
  - Expressivity of gadget set increases

- In 2 of 21 scenarios, no benefit to debloating was observed.

- Only 14 scenarios can be said to have benefitted from debloating.

Case Study: Security does not improve monotonically

- Debloating scenario libcurl (C) resulted in a number of negative side effects:
  - Increased the gadget set expressivity with respect to ASLR-Proof practical ROP exploits
  - Increased the gadget set expressivity with respect to simple Turing-completeness
  - Increased the number of system call gadgets
- GSA results from different libcurl debloating scenarios suggests these effects might be mitigated by removing fewer features.
- After modifying the debloating specification, building the new variant, and re-analyzing, the new results were largely an improvement, as expected:
  - Reduced gadget set expressivity with respect to ASLR-proof practical ROP exploits
  - No change in gadget set expressivity with respect to simple Turing-completeness
  - Decreased the number of system call gadgets

Summary of Key Takeaways

- The relationship between software debloating and software security is complicated.
  - Debloating can fail to improve security, or even make it worse.
  - Debloating for security is not like debloating for performance – debloating more does not necessarily produce better results.
- Measuring the reduction in gadget count is insufficient to make claims of improved security.
  - It can hide negative effects of debloating such as gadget introduction.
  - It is not directly related to more important measures such as availability of special purpose gadgets and gadget set expressivity.
- Debloating to improve security is possible, but not as easy as it looks.
  - It requires deep and multi-faceted analysis to determine the effect debloating had on security.
  - It may require multiple iterations to get it right.

Future Work

We do not claim that our proposed metrics are comprehensive.

We hope this work spurs further discussion about useful security metrics for debloating.

GSA is available in Github at:

- https://github.com/michaelbrownuc/GadgetSetAnalyzer

Recent Updates:

- Now uses freely available expressivity scanner
- Supports Gadget Locality metric for measuring moving target defense

Questions / Discussion

References

1. QUACH, A., ERINFOLAMI, R., DEMICCO, D., AND PRAKASH, A. A multi-OS cross-layer study of bloating in user programs, kernel, and managed execution environments. In *The 2017 Workshop on Forming an Ecosystem Around Software Transformation (FEAST)* (2017).
2. LEE, W., HEO, K., PASHAKHANLOO, P., AND NAIK, M. Effective Program Debloating via Reinforcement Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2018).
3. SHARIF, H., ABUBAKAR, M., GEHANI, A., AND ZAFFAR, F. TRIMMER: Application specialization for code debloating. In *Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)* (2018).
4. CHEN, Y., SUN, S., LAN, T., AND VENKATARAMANI, G. TOSS: Tailoring online server systems through binary feature customization. In *The 2018 Workshop on Forming an Ecosystem Around Software Transformation (FEAST)* (2018).
5. QUACH, A., PRAKASH, A., AND YAN, L. Debloating software through piece-wise compilation and loading. In *Proceedings of the 27th USENIX Security Symposium* (2018).
6. SALWAN, J. ROPgadget: Gadgets finder and auto-roper, 2011. http://shell-storm.org/project /ROPgadget/
7. SHACHAM, H. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In *Proceedings of 14th ACM conference on Computer and Communications Security (CCS)* (2007).
8. Bletsch, T., Jiang, X., Freeh, V.W., and Liang, Z. Jump-oriented programming: a new class of code-reuse attack. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS)* (2011)
9. HOMESCU, A., STEWART, M., LARSEN, P., BRUNTHALER, S., AND FRANZ, M. Microgadgets: size does matter in turing-complete return-oriented programming. In *Proceedings of the 6th USENIX conference on offensive technologies (WOOT)* (2012).