Semi-automated Feature-Debloating of Binary Software*

Dr. Kevin W. Hamlen
Eugene McDermott Professor
Computer Science Department
The University of Texas at Dallas

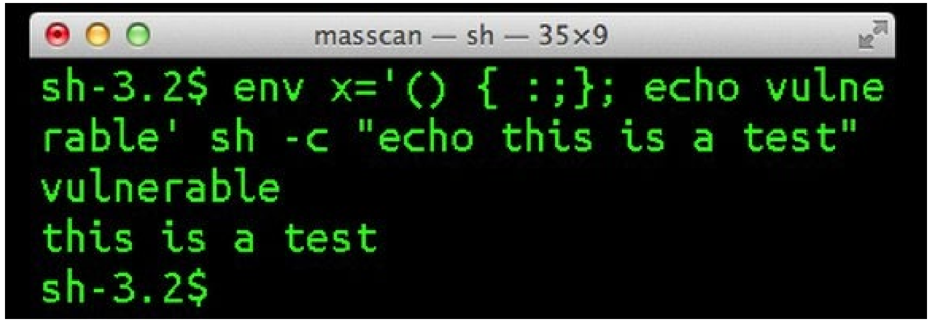
ONR TPCP Software Security Summer School (SSSS)
August 3, 2020

Publication: Masoud Ghaffarinia & Kevin W. Hamlen, "Binary Control-flow Trimming." In *Proc. ACM CCS* 2019.

*supported in part by ONR Award N00014-17-1-2995, NSF Award #1513704, and an endowment from the Eugene McDermott family.

Binary Control-flow Trimming

- **Objective:** Erase ("debloat") unwanted/unneeded features in binary software without the aid of source code
- **Motivating Example:** Linux Bash + Shellshock



- Discovered September 2014
- Bash shells execute certain environment variable texts as code(!!)
- Allows attackers to remote-compromise most Linux systems
- Window of vulnerability: 25 years(!!)
- Probably NOT originally a bug!
  - introduced in 1989 to facilitate function-import into child shells
  - never clearly documented, eventually forgotten

Research Challenges

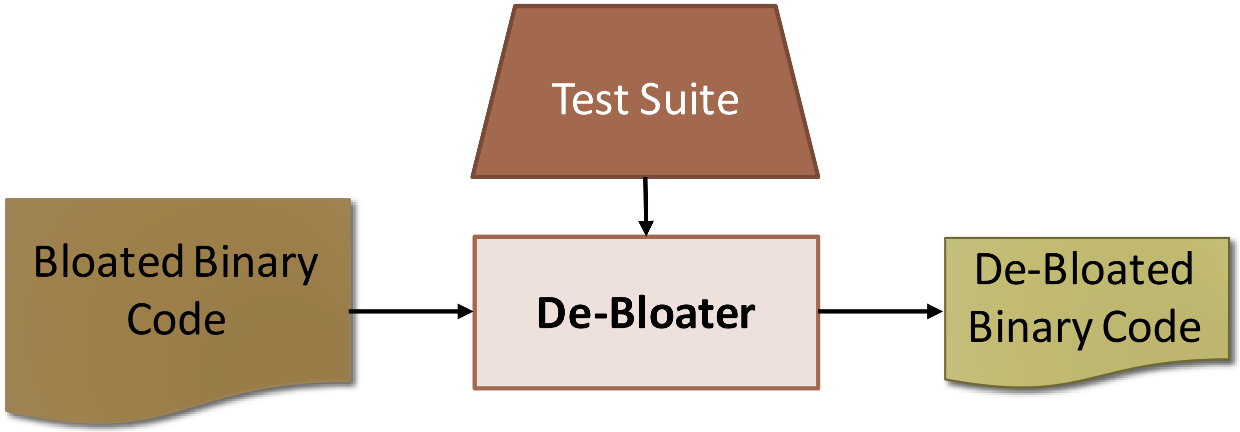
- Can we automatically erase unneeded (risky) functionalities from binary software?
  - Admins might not even know that the undesired functionality exists, and therefore *cannot necessarily demonstrate bugs/vulnerabilities.*
  - Demonstration of desired functionalities will usually be incomplete.
    - large input spaces (e.g., unbounded streams of network packets)
  - No assumptions about code design/provenance
    - arbitrary source languages
    - arbitrary compilation toolchains
    - simplifying assumption: not obfuscated (we can at least disassemble it)
- Can we do so without introducing significant inefficiencies?
  - no virtualization layers introduced
  - "debloated" code should be runnable on bare hardware

Basic Workflow

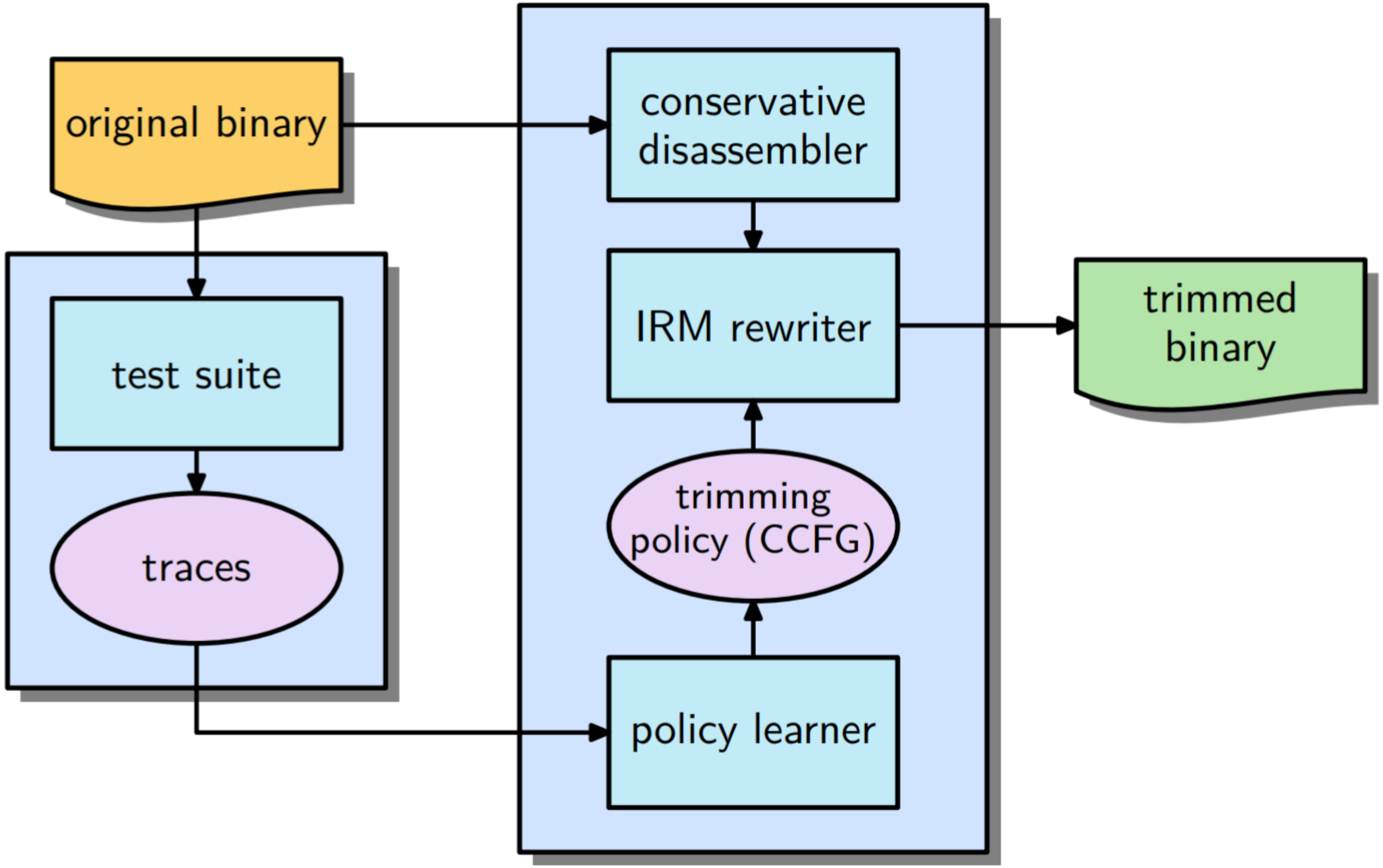
(1) Demonstrate representative desired functionalities by running the target software on various inputs in an emulator/VM.

(2) Submit resulting logs along with original binary code to de-bloater.

(3) If resulting de-bloated binary is unsatisfactory (e.g., needed functionalities missing), then repeat with more/better tests.



Binary Control-flow Trimming Architecture



Stepwise Usage

1. CCFI-protect binary with a permit-all policy

- `rewriter-makeout.py --learn --target $BCFT_TARGET_BINARY …`

2. run new binary in emulator (PIN) on training inputs

- `pin -i … -o … -- $PROGRAM $ARGS`

3. learn a CCFI policy from the traces logged by the emulator

- `learner.py $PROGRAM_TRACES_DIR`

4. replace the permit-all policy with the learned policy

- `rewriter-makeout.py --policy $POLICY_FILE --target $BCFT_BINARY`

Experiments and Evaluations

- Performance:
  - SPEC CPU Benchmark.
  - Lighttpd, Nginx web-servers.
  - Proftpd, pureftpd, vsftpd ftp-servers.
- Test-suite for accuracy and security:

Program	Test Suite	Debloated Functionalities
GCC	Its own source code.	-m32 (accuracy)
Ftp-servers	Random files mixed with commands (e.g. rm).	SITE, DELETE (security, accuracy)
Browsers	Quantcast top 475K URLs.	Incognito, cookies add/delete(accuracy)
ImageMagic convert	Converting random jpgs to png.	resizing(accuracy)
Exim	Random emails to a specific address.	-ps (security), -oMs(accuracy)
Node.js	Java scrip code not using `serialize()`.	`serialize()`(security)

Vulnerabilities Removed

Successfully removed Shellshock vulnerability using only the pre-Shellshock test-suite shipped with bash.

Program	CVE numbers
Bash	CVE-2014-6271, -6277, -6278, -7169
ImageMagic	CVE-2016-3714, -3715, -3716, -3717, -3718
Proftpd	CVE-2015-3306
Node.js	CVE-2017-5941
Exim	CVE-2016-1531

Limitations and Scope

- **<u>DON'T</u>** use this if…
  - … you have full source code and can recompile all system components.
  - … you want to shrink the software's memory image.
  - … it is difficult/impossible to demonstrate all critical functionalities.
    - (In future research we want to relax this restriction.)
- **<u>DO</u>** use this if…
  - … you don't have or don't trust some/all of the source code for the software.
  - … the software has *no formal specification* of correctness/security.
  - … you have no developer cooperation for finding/fixing bugs/features.
  - … you want to run the code natively (no VM).

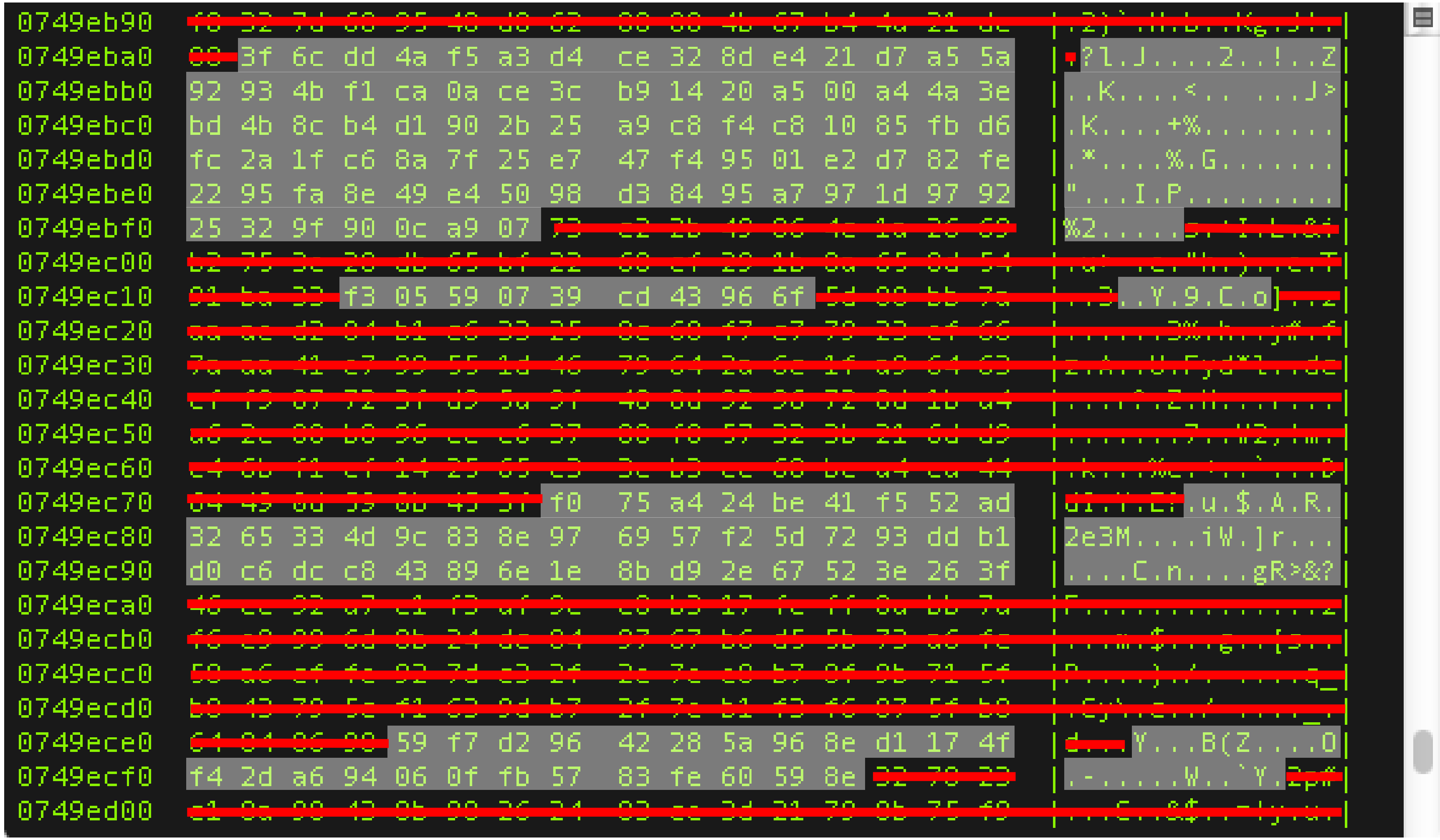
Obvious Approach: Code Byte Erasure

```
0749eb90  f0 32 7d 60 95 48 d0 62  08 80 4b 67 b4 4a 21 dc  |.2}`.H.b..Kg.J!.|
0749eba0  80 3f 6c dd 4a f5 a3 d4  ce 32 8d e4 21 d7 a5 5a  |.?l.J....2..!..Z|
0749ebb0  92 93 4b f1 ca 0a ce 3c  b9 14 20 a5 00 a4 4a 3e  |..K....<.. ...J>|
0749ebc0  bd 4b 8c b4 d1 90 2b 25  a9 c8 f4 c8 10 85 fb d6  |.K....+%........|
0749ebd0  fc 2a 1f c6 8a 7f 25 e7  47 f4 95 01 e2 d7 82 fe  |.*....%.G.......|
0749ebe0  22 95 fa 8e 49 e4 50 98  d3 84 95 a7 97 1d 97 92  |"...I.P.........|
0749ebf0  25 32 9f 90 0c a9 07 73  c2 2b 49 06 4c 1a 26 69  |%2.....s.+I.L.&i|
0749ec00  b2 75 3e 20 db 65 bf 22  68 cf 29 1b 8a 65 8d 54  |.u> .e."h.)..e.T|
0749ec10  91 ba 33 f3 05 59 07 39  cd 43 96 6f 5d 88 bb 7a  |..3..Y.9.C.o]..z|
0749ec20  aa ae d2 04 b1 c6 33 25  8c 68 f7 c7 79 23 ef 66  |......3%.h..y#.f|
0749ec30  7a aa 41 e7 99 55 1d 46  79 64 2a 6c 1f a9 64 63  |z.A..U.Fyd*l..dc|
0749ec40  ef f9 87 72 3f d9 5a 9f  48 0d 92 96 72 0d 1b a4  |...r?.Z.H...r...|
0749ec50  a6 2e 08 b0 96 cc e6 37  88 f0 57 32 3b 21 6d d9  |.......7..W2;!m.|
0749ec60  e4 6b f1 ef 14 25 65 e3  3c b3 ee 60 bc a4 ea 44  |.k...%e.<..`...D|
0749ec70  64 49 0d 59 0b 45 3f f0  75 a4 24 be 41 f5 52 ad  |dI.Y.E?.u.$.A.R.|
0749ec80  32 65 33 4d 9c 83 8e 97  69 57 f2 5d 72 93 dd b1  |2e3M....iW.]r...|
0749ec90  d0 c6 dc c8 43 89 6e 1e  8b d9 2e 67 52 3e 26 3f  |....C.n....gR>&?|
0749eca0  46 cc 92 a7 e1 f3 af 9c  c8 b3 17 fe ff 8a bb 7a  |F..............z|
0749ecb0  f6 e9 99 6d 8b 24 dc 84  97 67 b6 d5 5b 73 a6 fc  |...m.$...g..[s..|
0749ecc0  50 a6 cf fe 92 7d c3 2f  2e 7e e8 b7 8f 9b 71 5f  |P....}./.~....q_|
0749ecd0  b0 43 79 5c f1 63 9d b7  2f 7e b1 f3 f6 87 5f b0  |.Cy\.c../~...._.|
0749ece0  64 84 86 98 59 f7 d2 96  42 28 5a 96 8e d1 17 4f  |d...Y...B(Z....O|
0749ecf0  f4 2d a6 94 06 0f fb 57  83 fe 60 59 8e 32 70 23  |.-.....W..`Y.2p#|
0749ed00  c1 8a 98 43 0b 90 26 24  03 ce 3d 21 79 0b 75 f9  |...C..&$..=!y.u.|
```

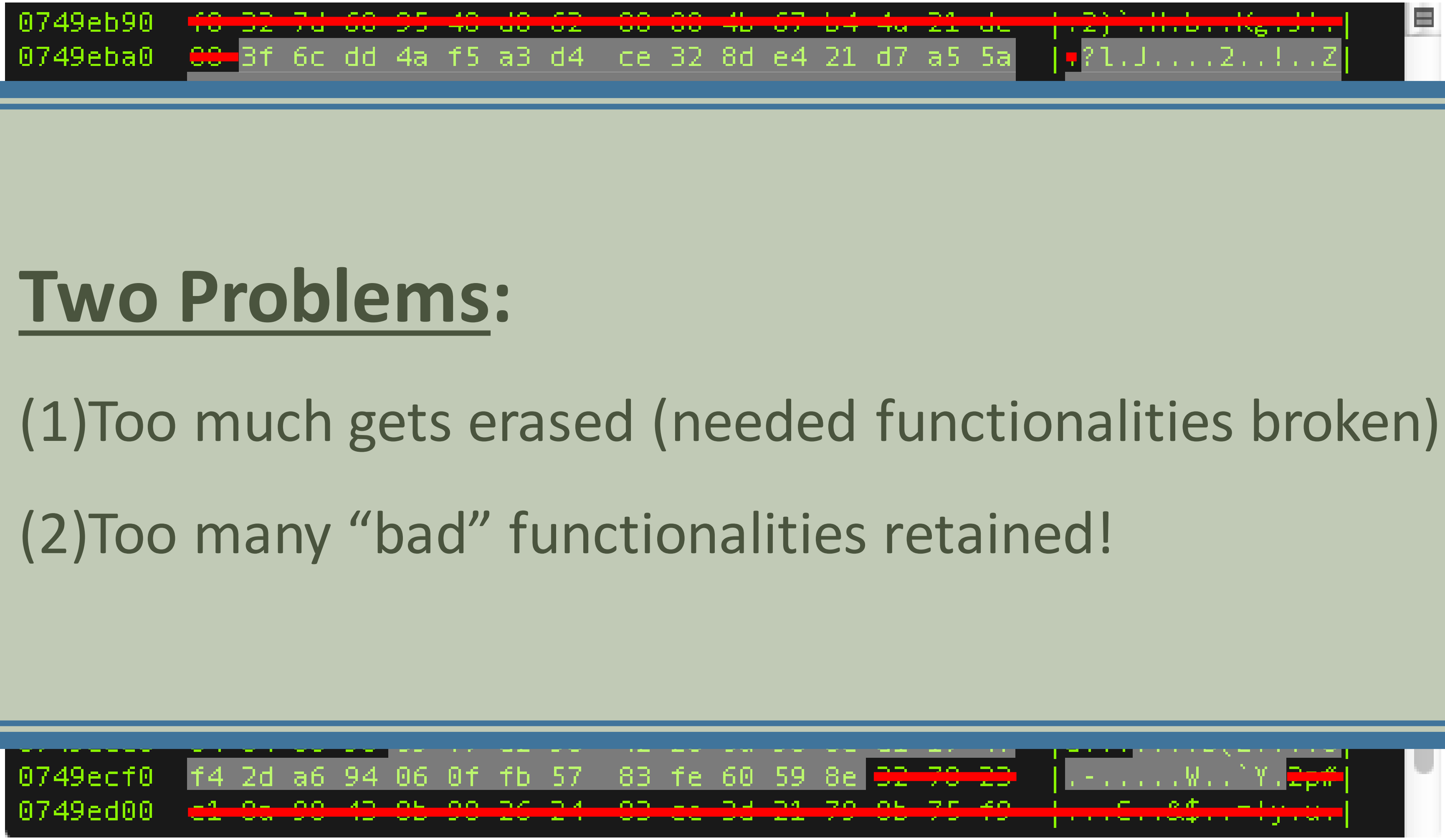
Obvious Approach: Code Byte Erasure

```
0749eb90  f0 32 7d 60 95 48 d0 62  08 80 4b 67 b4 4a 21 dc  |.2}`.H.b..Kg.J!.|
0749eba0  80 3f 6c dd 4a f5 a3 d4  ce 32 8d e4 21 d7 a5 5a  |.?l.J....2..!..Z|
0749ebb0  92 93 4b f1 ca 0a ce 3c  b9 14 20 a5 00 a4 4a 3e  |..K....<.. ...J>|
0749ebc0  bd 4b 8c b4 d1 90 2b 25  a9 c8 f4 c8 10 85 fb d6  |.K....+%........|
0749ebd0  fc 2a 1f c6 8a 7f 25 e7  47 f4 95 01 e2 d7 82 fe  |.*....%.G.......|
0749ebe0  22 95 fa 8e 49 e4 50 98  d3 84 95 a7 97 1d 97 92  |"...I.P.........|
0749ebf0  25 32 9f 90 0c a9 07 73  c2 2b 49 06 4c 1a 26 69  |%2.....s.+I.L.&i|
0749ec00  b2 75 3e 20 db 65 bf 22  68 cf 29 1b 8a 65 8d 54  |.u> .e."h.)..e.T|
0749ec10  91 ba 33 f3 05 59 07 39  cd 43 96 6f 5d 88 bb 7a  |..3..Y.9.C.o]..z|
0749ec20  aa ae d2 04 b1 c6 33 25  8c 68 f7 c7 79 23 ef 66  |......3%.h..y#.f|
0749ec30  7a aa 41 e7 99 55 1d 46  79 64 2a 6c 1f a9 64 63  |z.A..U.Fyd*l..dc|
0749ec40  ef f9 87 72 3f d9 5a 9f  48 0d 92 96 72 0d 1b a4  |...r?.Z.H...r...|
0749ec50  a6 2e 08 b0 96 cc e6 37  88 f0 57 32 3b 21 6d d9  |.......7..W2;!m.|
0749ec60  e4 6b f1 ef 14 25 65 e3  3c b3 ee 60 bc a4 ea 44  |.k...%e.<..`...D|
0749ec70  64 49 0d 59 0b 45 3f f0  75 a4 24 be 41 f5 52 ad  |dI.Y.E?.u.$.A.R.|
0749ec80  32 65 33 4d 9c 83 8e 97  69 57 f2 5d 72 93 dd b1  |2e3M....iW.]r...|
0749ec90  d0 c6 dc c8 43 89 6e 1e  8b d9 2e 67 52 3e 26 3f  |....C.n....gR>&?|
0749eca0  46 cc 92 a7 e1 f3 af 9c  c8 b3 17 fe ff 8a bb 7a  |F..............z|
0749ecb0  f6 e9 99 6d 8b 24 dc 84  97 67 b6 d5 5b 73 a6 fc  |...m.$...g..[s..|
0749ecc0  50 a6 cf fe 92 7d c3 2f  2e 7e e8 b7 8f 9b 71 5f  |P....}./.~....q_|
0749ecd0  b0 43 79 5c f1 63 9d b7  2f 7e b1 f3 f6 87 5f b0  |.Cy\.c../~...._.|
0749ece0  64 84 86 98 59 f7 d2 96  42 28 5a 96 8e d1 17 4f  |d...Y...B(Z....O|
0749ecf0  f4 2d a6 94 06 0f fb 57  83 fe 60 59 8e 32 70 23  |.-.....W..`Y.2p#|
0749ed00  c1 8a 98 43 0b 90 26 24  03 ce 3d 21 79 0b 75 f9  |...C..&$..=!y.u.|
```

Obvious Approach: Code Byte Erasure



Obvious Approach: Code Byte Erasure



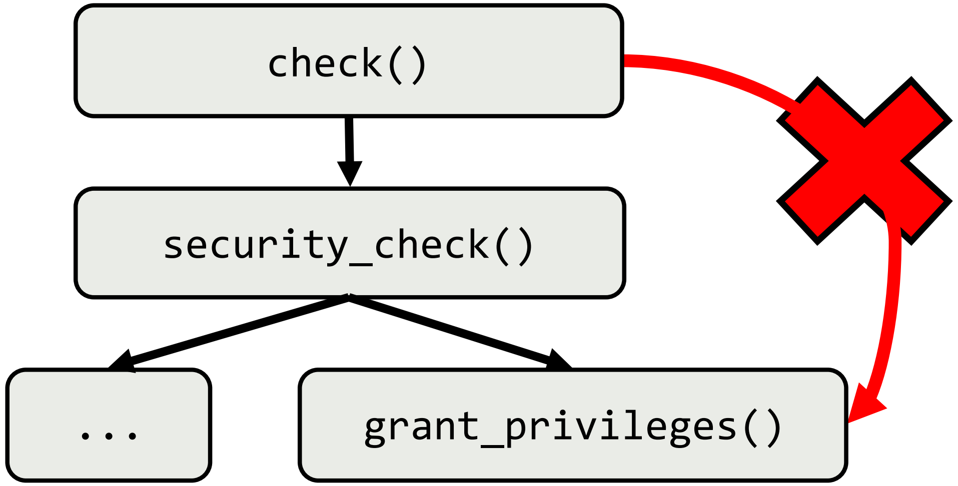
<u>Two Problems</u>:

(1)Too much gets erased (needed functionalities broken)

(2)Too many “bad” functionalities retained!

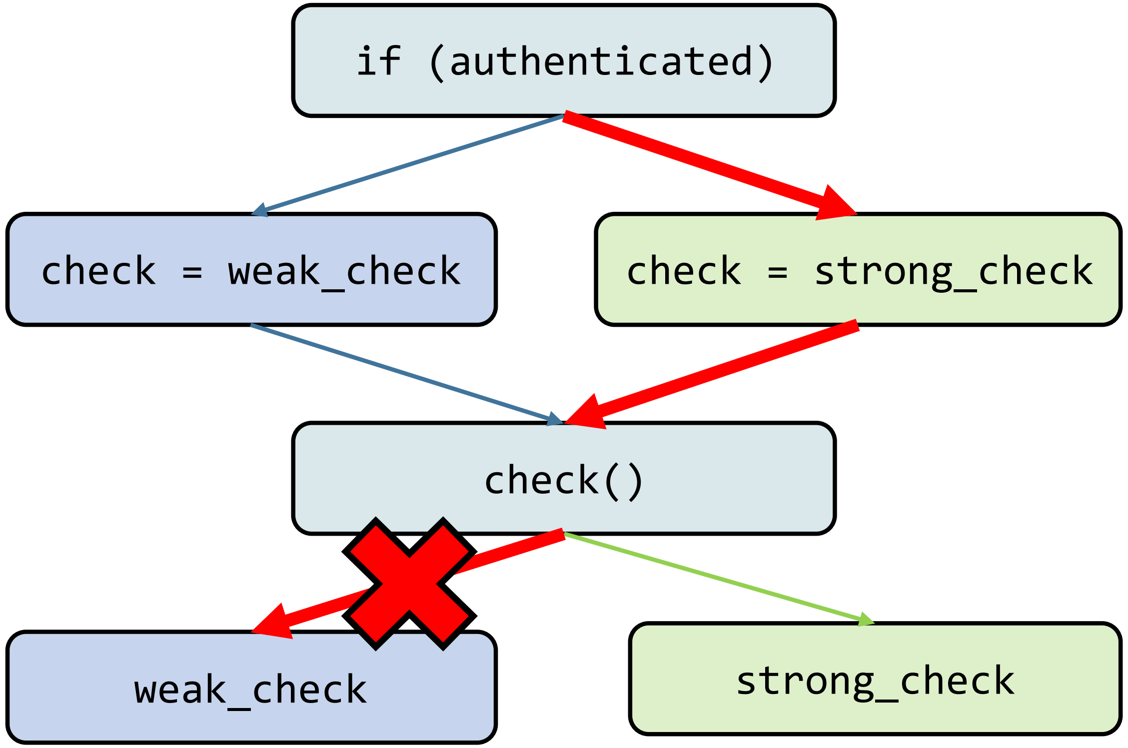
Code Erasure vs. Edge Erasure

```
void access_database() {

    bool (*check)(void);
    char vul_buf[N];

    check = &security_check;

    ...


    scanf("%s", vul_buf);

    if (check()) {
        grant_privileges();
    }
}
```



Edge Erasure vs. Flow Erasure

```
void access_database() {

      bool (*check)(void);
      char vul_buf[N];

      if (authenticated)
            check = weak_check;
      else
            check = strong_check;

      scanf("%s", vul_buf);

      if (check()) {
            grant_privileges();
      }
}
```

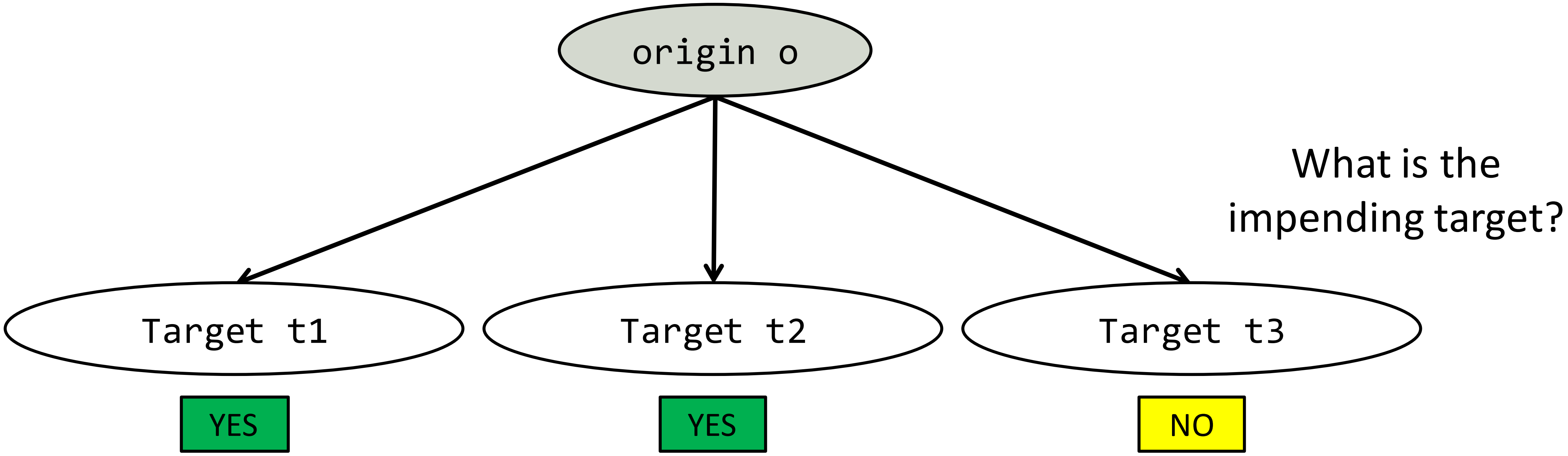


Contextual Control-flow Integrity (CCFI)

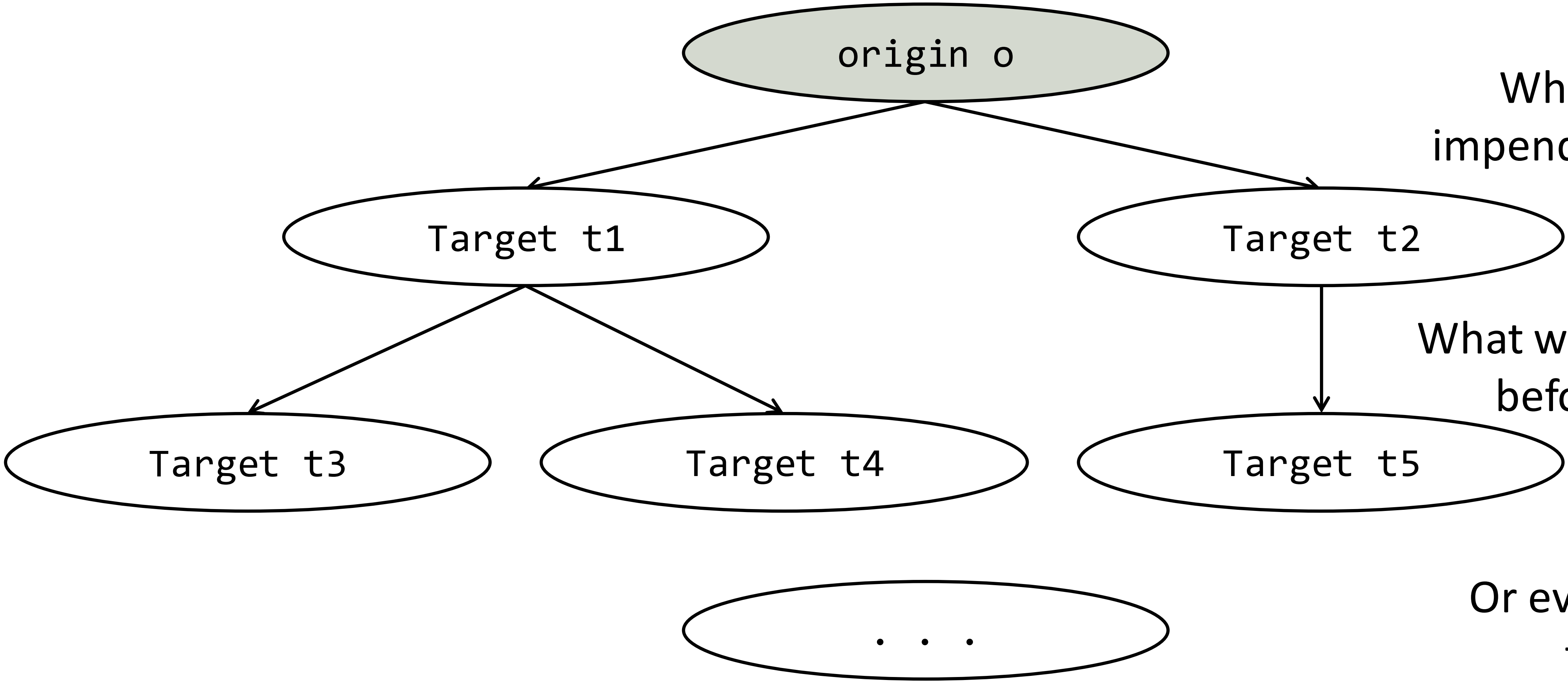
- Basic implementation strategy
  - Replace each jump/branch/call instruction in the original code with a *check-then-jump* sequence
  - The "check" code updates and consults a saved *context history* of previous jumps.
- Requirements
  - ALL jump/branch/calls must be replaced
  - saved context history must be protected from attacker modification
- Prior work
  - non-contextual CFI enforcement is well-established
  - contextual CFI is very hard to implement efficiently
    - PathArmor [Van Der Veen et al.; USENIX Sec '15]: only checks system API calls, has high overhead
- Main challenge #1: How to learn a CCFI policy without a spec?
- Main challenge #2: How to enforce such fine-grained CCFI efficiently?

Learning CFG Policy

- Decision Trees at every branch site.



Learning Contextual CFG Policy



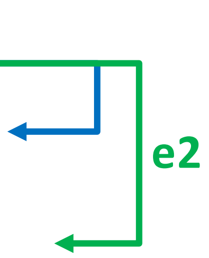
What is the impending target?

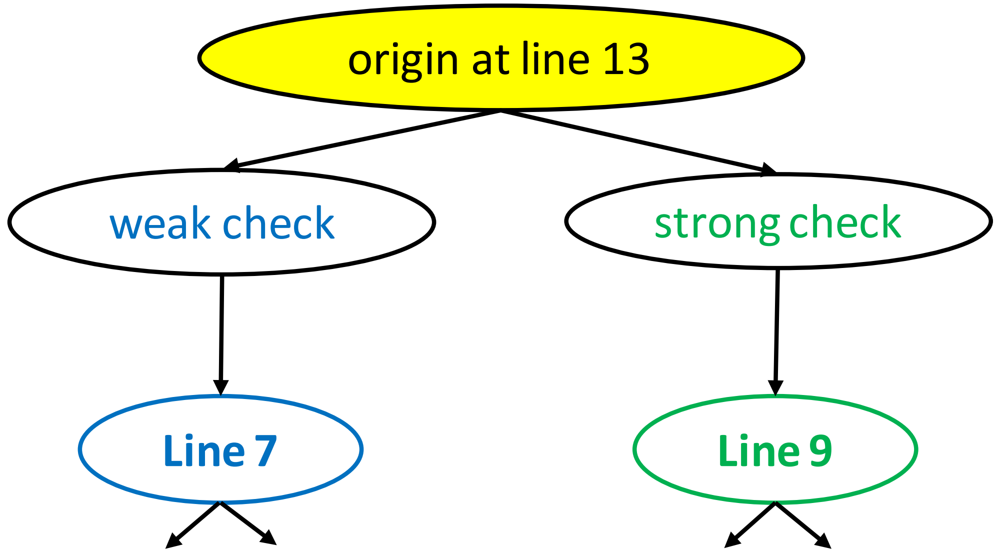
What was the target before that?

Or even before that?

Contextual CFG Trees

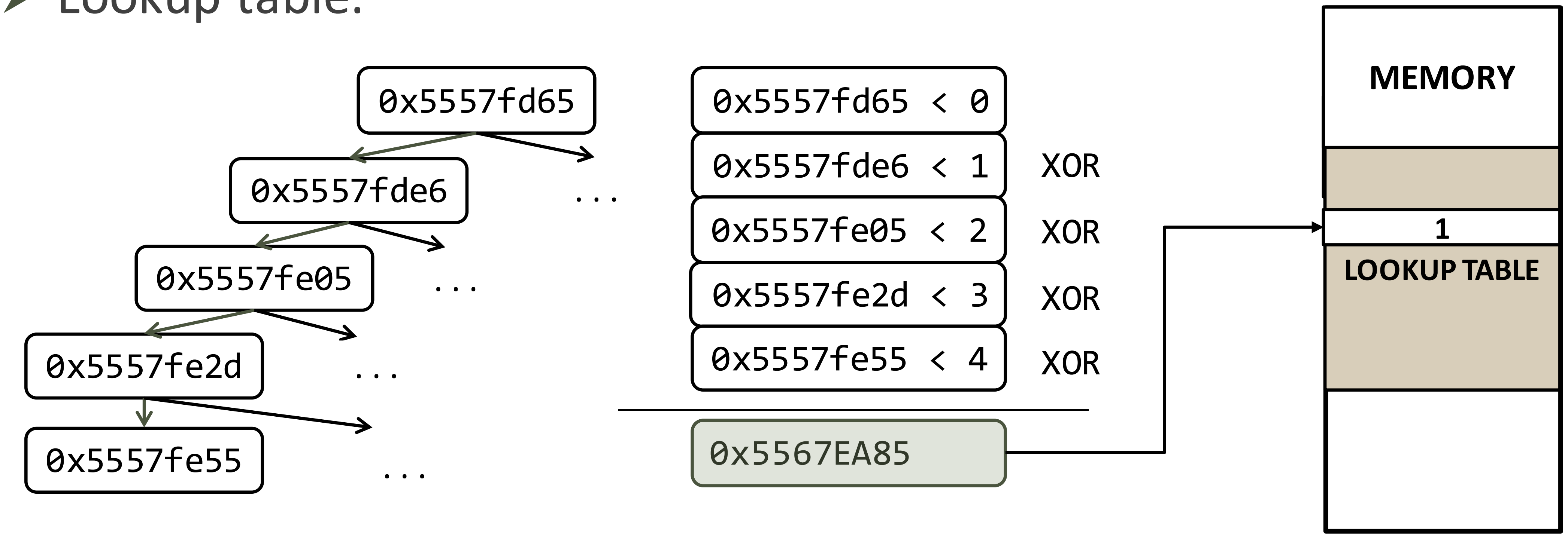
```
1   void access_database() {
2
3       bool (*check)(void);
4       char vul_buf[N];
5
6       if (authenticated)
7           check = weak_check;
8       else
9           check = strong_check;
10
11      scanf("%s", vul_buf);
12
13      if (check()) {
14          grant_privileges();
15      }
16  }
```





Policy Representation

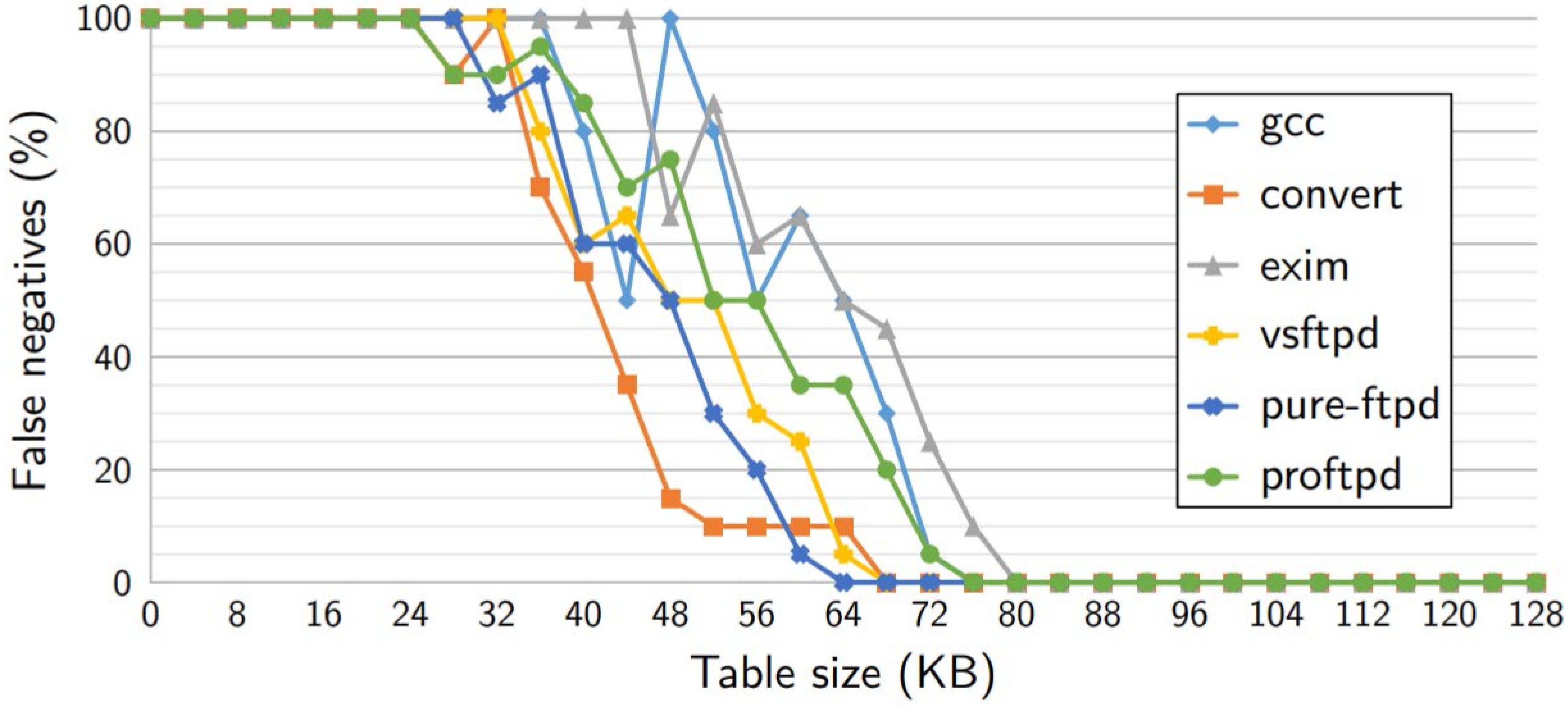
- Lookup table.



$$hash(\chi) = \bigoplus_{i=1}^{|\chi|} ((\pi_2 \chi_i) \ll (|\chi| - i)s) \qquad hash(\chi e) = (hash(\chi) \ll s) \oplus (\pi_2 e)$$

Hash Table Sizes

A table of size n B can whitelist $8n$ contexts.



Guard Checks

Description	Original code	Rewritten Code
Conditional Jumps	*jcc* *l*	`call` *jcc*`_fall` `.quad` *l*
Indirect calls	`call` *r*/[*m*]	`mov` *r*/[*m*]`, %rax` `call indirect_call`

Indirect Jumps	`jmp` *r*/[*m*]	`mov %rax, -16(%rsp)`
		`mov` *r*/[*m*]`, %rax`
		`call indirect_jump`

Variable Returns	`ret` *n*	`pop %rdx`
		`lea` *n*`(%rsp), %rsp`
		`push %rdx`
		`jmp return`

Returns	`ret`	`mov (%rsp), %rdx` `jmp return`

Label	Assembly Code

`indirect_jump:`	`push %rax`
	`common-guard`
	`mov -8(%rsp), %rax`
	`ret`

`indirect_call:`	`push %rax`
	`common-guard`
	`ret`

`return:`	`common-guard` `ret`
jcc`_fall:`	*jcc* `jump_l` `jmp fall_l`
jcc`_back:`	*jcc* `jump_l` `jmp back_l`

`jump_l:`	`xchg (%rsp), %rax`
	`mov (%rax), %rax`
	`jmp condition_jump`

`fall_l:`	`xchg (%rsp), %rax`
	`lea 8(%rax), %rax`
	`jmp condition_jump`

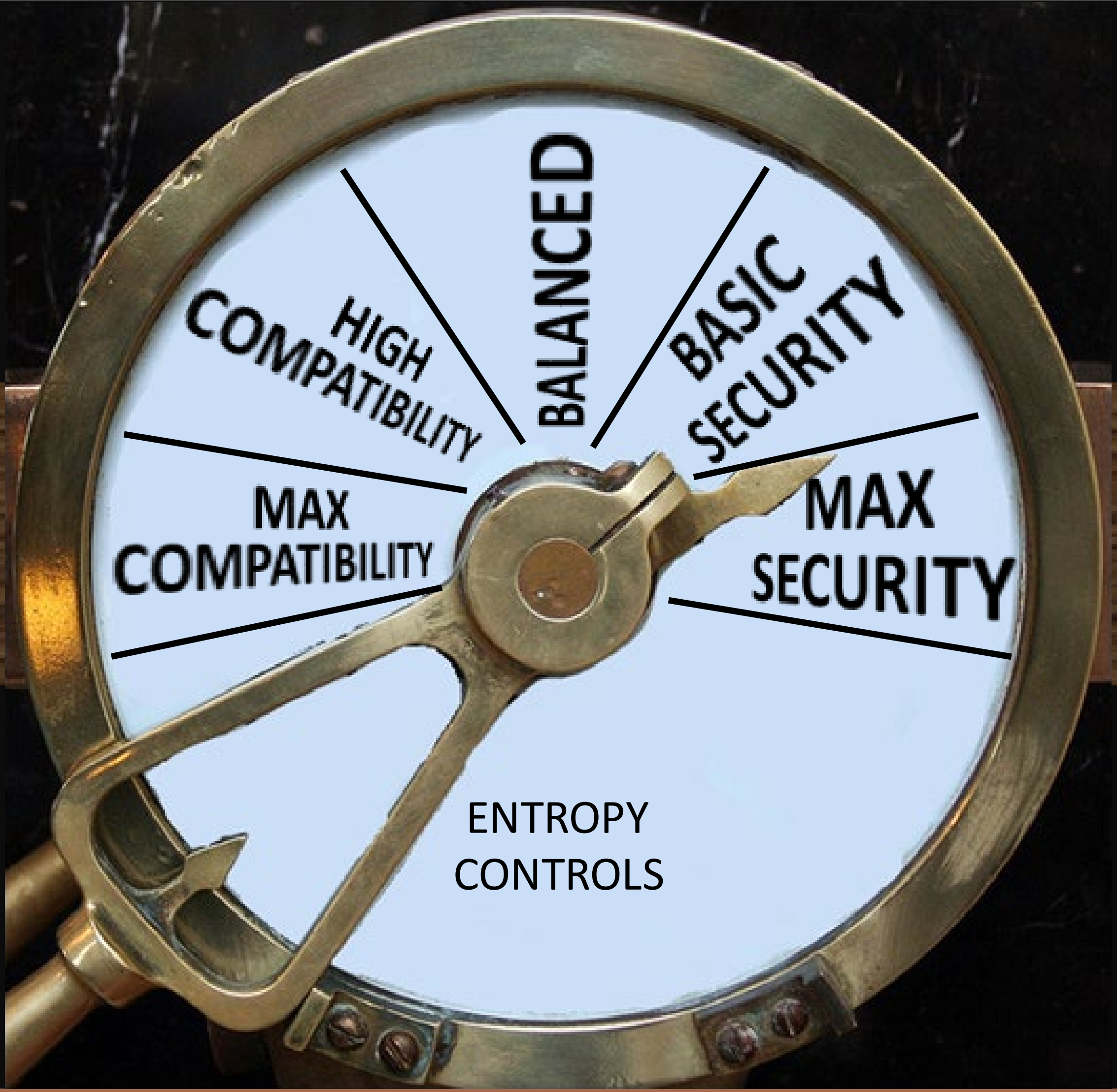
`back_l:`	`xchg (%rsp), %rax`
	`lea 8(%rax), %rax`
	`xchg (%rsp), %rax`
	`ret`

`condition_jump:`	`push %rax`
	`common-guard`
	`pop %rax`
	`xchg (%rsp), %rax`
	`ret`

Context Protection with Wide Registers

Guard Name	Guard Code: Legacy-mode		Guard Code: SHA-extension	
before-check	1:movd	r, %xmm11	1:movd	r, %xmm11
	2:psubd	%xmm12, %xmm11	2:psubd	%xmm12, %xmm11
			3:sha1msg1	%xmm14, %xmm13
			4:sha1msg2	%xmm13, %xmm13
			5:pslrdq	$4, %xmm13
	3:pxor	%xmm11, %xmm13	6:pxor	%xmm11, %xmm13
check	4:movd	%xmm13, r	7:movd	%xmm13, r
	5:and	$(max\_hash - 1)$, r	8:and	$(max\_hash - 1)$, r
	6:bt	r, (HASH_TABLE)	9:bt	r, (HASH_TABLE)
	7:jnb	TRAP	10:jnb	TRAP
after-check	8:pextrd	$3, %xmm14, r	11:pslldq	$4, %xmm14
	9:pslldq	$4, %xmm14	12:psllw	$1, %xmm14
	10:pxor	%xmm11, %xmm14	13:pxor	%xmm11, %xmm14
	11:movd	r, %xmm11		
	12:pxor	%xmm11, %xmm13		
	13:pslld	$1, %xmm13		
	14:pslld	$1, %xmm14		

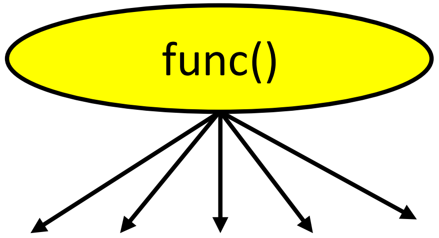
Tuning Policy Strictness



Decision Trees and Entropy

- High entropy node = high uncertainty = incomplete testing

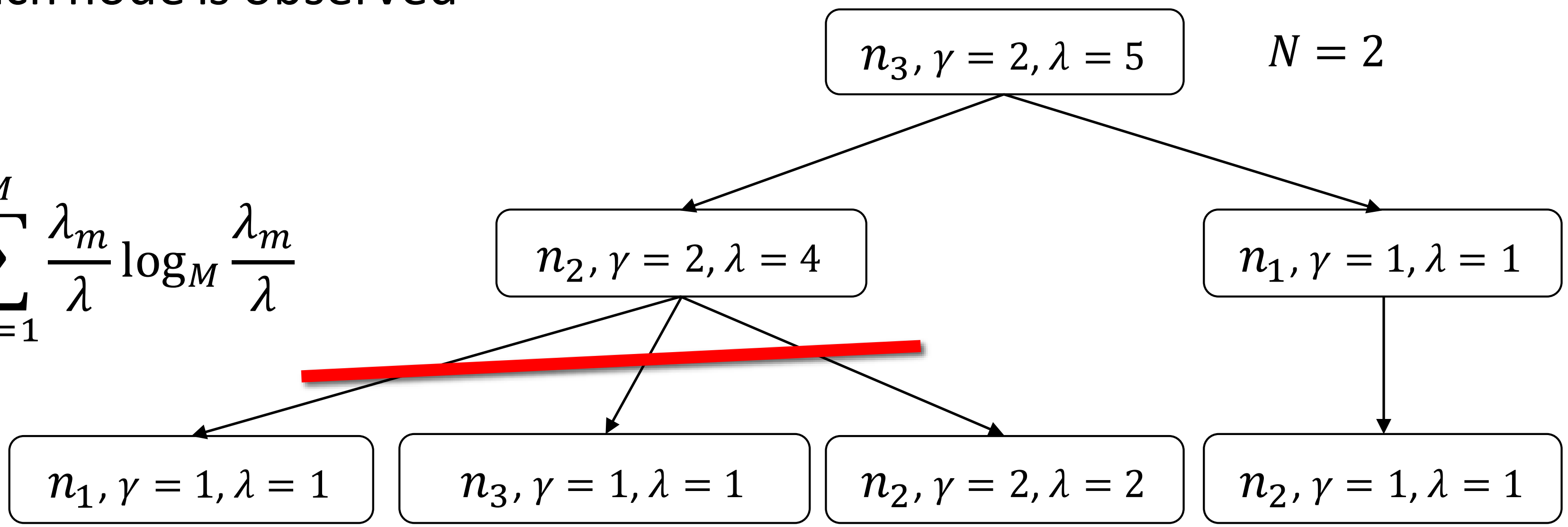
```
void dispatch(void (*func)()) {
    func();
    LOG();
}
```



Relaxing the policy

- **Relaxation philosophy:**
  - Relaxed policy is always as strict as non-contextual CFI.
  - Relaxations merely identify some context as irrelevant to the enforcement decision.
- **Parameters**
  - λ = # times the node observed in all traces
  - γ = # traces in which node is observed
  - N = total traces
  - M = # children

$$score(n) = \frac{\gamma}{N} \times -\frac{1}{\mathrm{M}^2} \sum_{m=1}^{M} \frac{\lambda_m}{\lambda} \log_M \frac{\lambda_m}{\lambda}$$

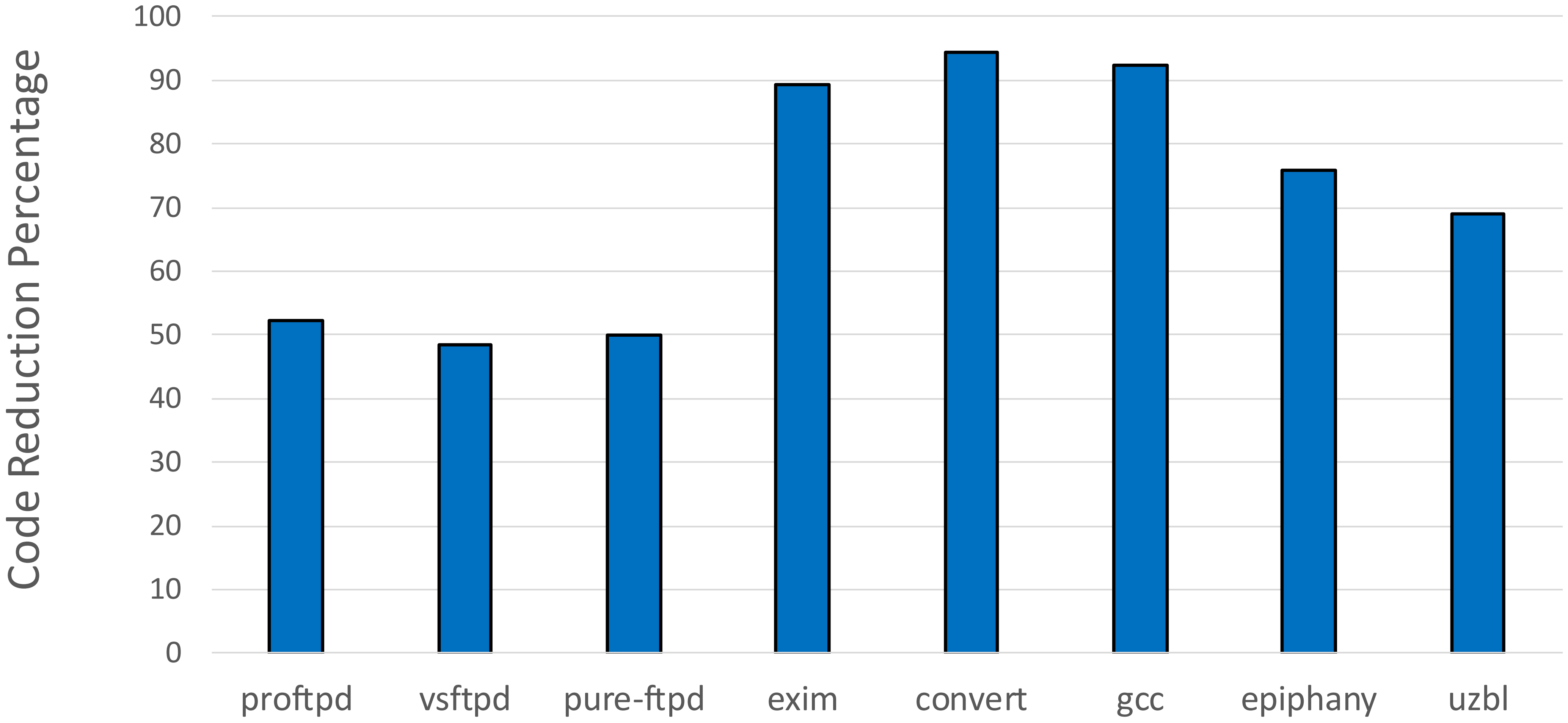


Accuracy

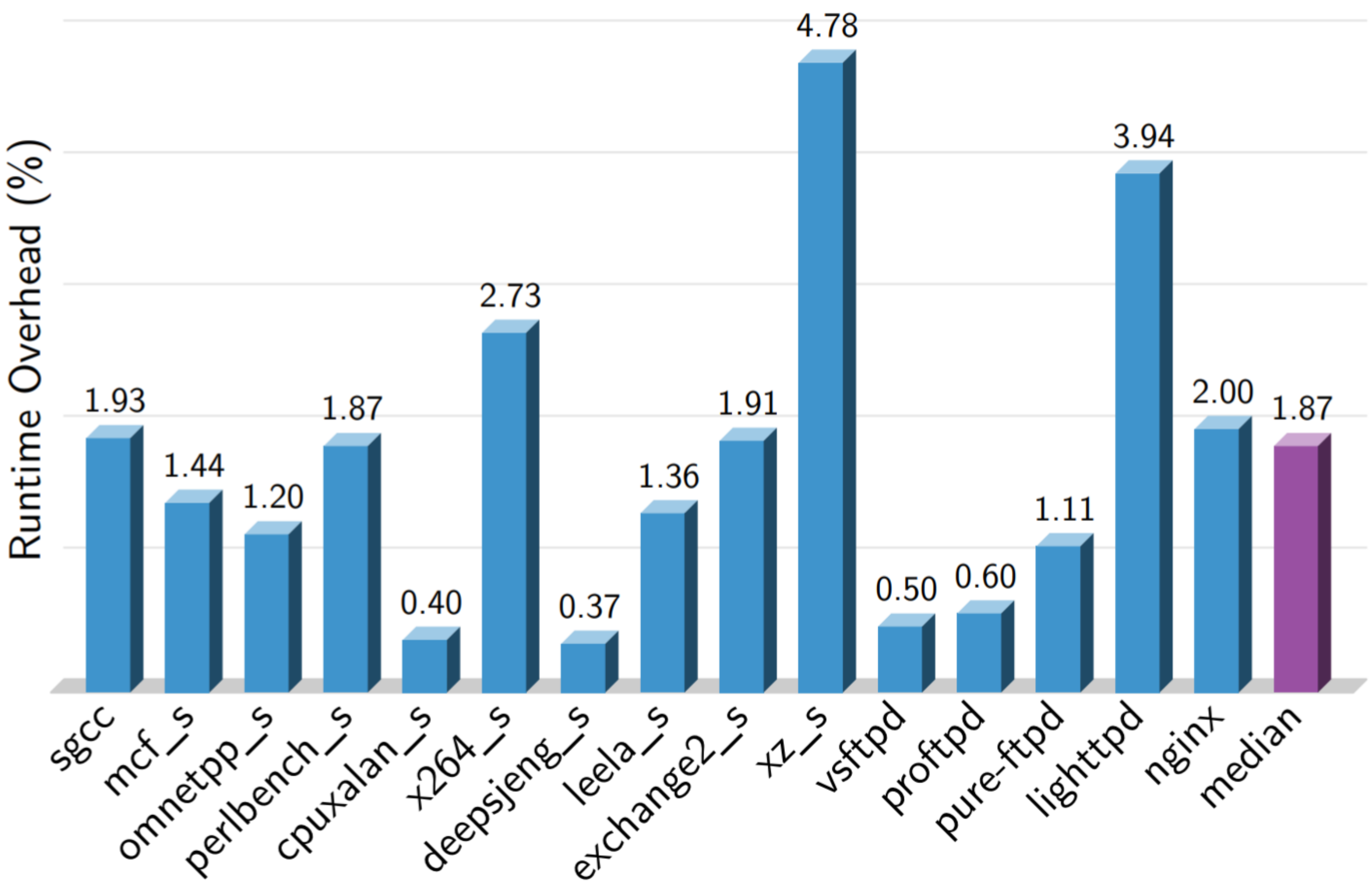
		Program											
		proftpd			**vsftpd**			**pure-ftpd**			**exim**		
	Sample Size	10	100	500	10	100	500	10	100	500	10	100	200
	t^*	0.48	0.37	0.00	0.38	0.23	0.00	0.41	0.28	0.00	0.25	0.53	0.00
FP	**t=0.00**	45.00	3.00	0.00	35.00	2.00	0.00	25.00	2.50	0.00	35.00	7.50	0.00
	t=0.25	30.00	1.50	0.00	25.00	1.50	0.00	25.00	1.50	0.00	15.00	1.00	0.00
	$t=t^*$	25.00	1.00	0.00	25.00	1.50	0.00	10.00	1.50	0.00	20.00	0.00	0.00
FN		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

		Program											
		epiphany				**uzbl**				**convert**			**gcc**
	Sample Size	10	100	500	1000	10	100	500	1000	10	100	200	10
	t^*	0.93	0.81	0.33	0.00	0.92	0.83	0.65	0.45	0.64	0.54	0.00	0.00
FP	**t=0.00**	85.00	40.00	8.70	0.00	90.00	50.50	10.70	4.30	20.00	2.50	0.00	0.00
	t=0.25	40.00	10.00	0.40	0.00	40.00	3.50	0.90	0.85	15.00	1.00	0.00	0.00
	$t=t^*$	0.00	6.50	0.30	0.00	30.00	2.50	0.60	0.35	10.00	0.00	0.00	0.00
FN		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Reachable Code Reduction



Run-time Overhead



CFI ≠ Debloating

- Policies enforced by prior CFI works:
  - Source-aware CFI solutions: CFG derived from source code semantics
  - Binary-only CFI solutions: Approximate the source CFG from binary semantics
  - Both approaches preserve <u>developer-intended</u>, consumer-unwanted edges.

- Prior contextual CFI solution:
  - PathArmor [Van Der Veen et al.; USENIX Security 2015]
    - Contextual checks only performed at system call sites
    - Insufficient granularity to debloat fine-grained code blocks from software
    - Performance overhead too high if applied to every branch instruction

Comparison with RAZOR [Qian et al. (USENIX'19)]

	RAZOR	Control-flow Trimming
Strategy	Heuristics applied to code structure and traces	Machine learning (decision trees)
Policy Expressiveness	Static CFI	Contextual CFI
Debloating rate	~71%	~71%
Performance Overhead	1.7%	1.9%

Conclusion

- Main achievements
  - Binary software debloating using incomplete test-suite and no source code
  - First fine-grained contextual CFI enforcement at every branch site with high performance (1.8% overhead)
- Challenges for Future Research / Transition
  - Highly interactive software (diverse traces) can create high training burden. Could couple with directed fuzzers to improve training effectiveness.
  - Training process automatically detects uncertainties and ambiguities. Feed this information back to (non-expert) users to help them refine the training?

THANK YOU

QUESTIONS?