Thread-Level Attack-Surface Reduction

Florian Rommel
rommel@sra.uni-hannover.de
Leibniz Universität Hannover

Christian Dietrich
christian.dietrich@tuhh.de
Hamburg University of Technology

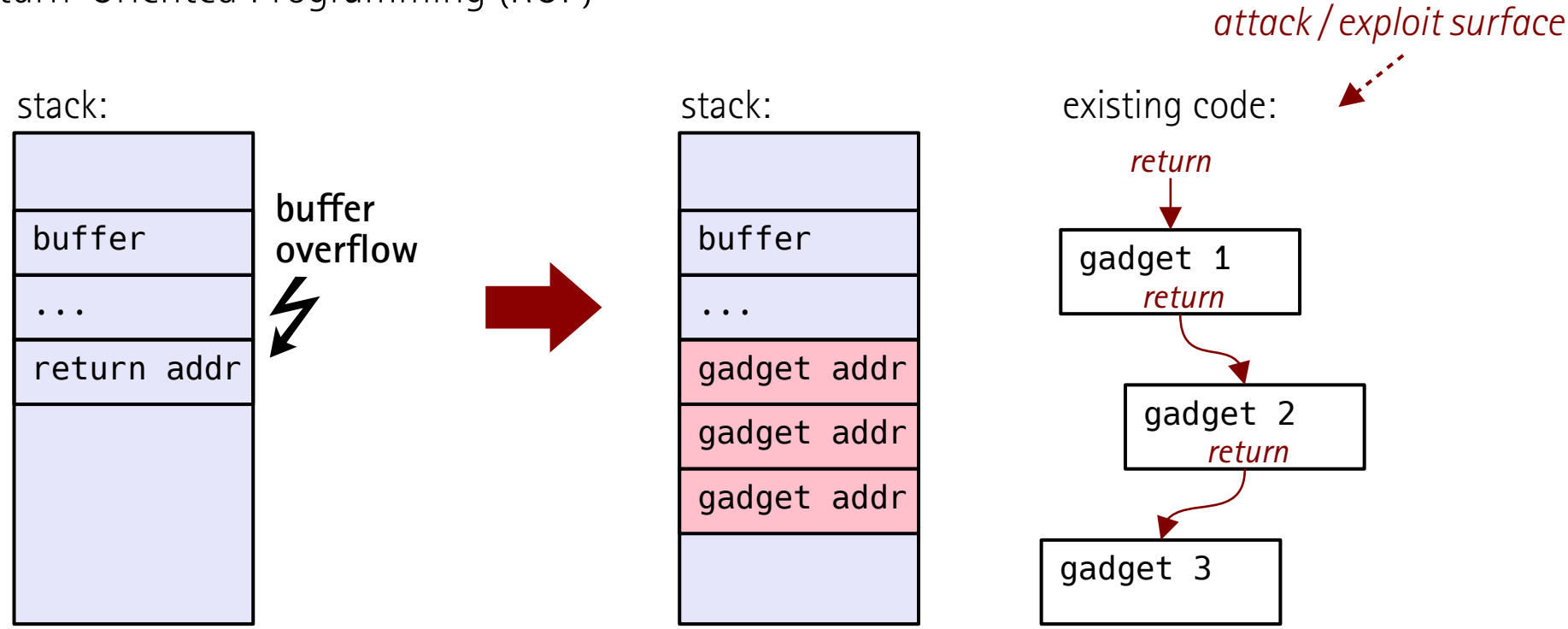
Andreas Ziegler
ziegler@cs.fau.de
Friedrich-Alexander-Universität Erlangen-Nürnberg

Illia Ostapyshyn
ostapyshyn@sra.uni-hannover.de
Leibniz Universität Hannover

Daniel Lohmann
lohmann@sra.uni-hannover.de
Leibniz Universität Hannover

Attack / Exploit Surface

Return-Oriented Programming (ROP)



Binary Debloating

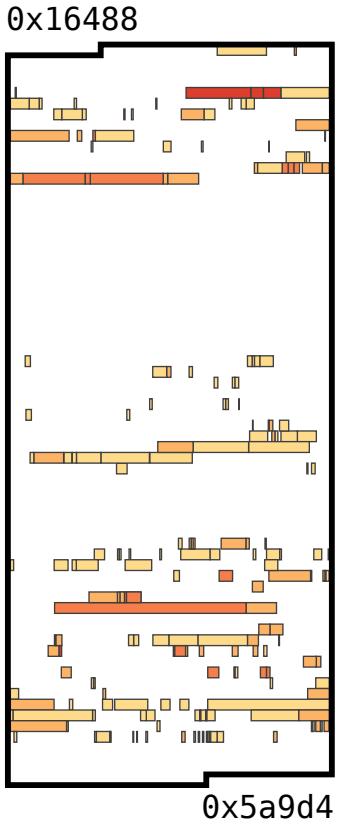
Remove unnecessary code from dynamic libraries

Why?

→ Smaller binaries

→ Reduced attack/exploit surface

Still a lot of code present (all potentially needed code from libs + whole main executable)



Ziegler et al. '19

"Honey, I Shrunk the ELFs: Lightweight Binary Tailoring of Shared Libraries."
ACM Transactions on Embedded Computing Systems

Use of musl libc functions by vsftpd

Executions

- unused
- 1 - 10
- ≤ 100
- ≤ 1000
- ≤ 10000

max 7068

Dynamic Debloating

Remove unnecessary code from running processes

Porter et al. '20
"BlankIt Library Debloating" PLDI

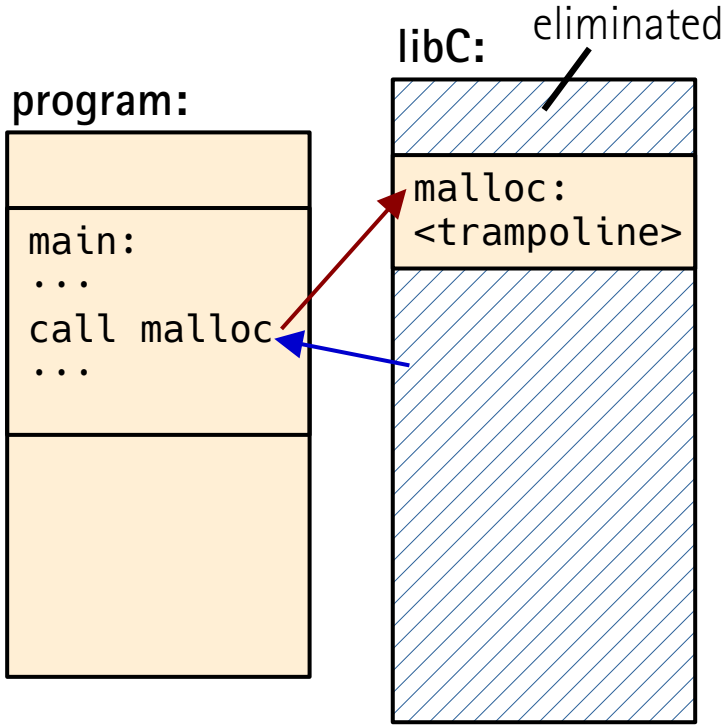
Why?

→ ~~Smaller binaries~~

→ Reduced attack/exploit surface

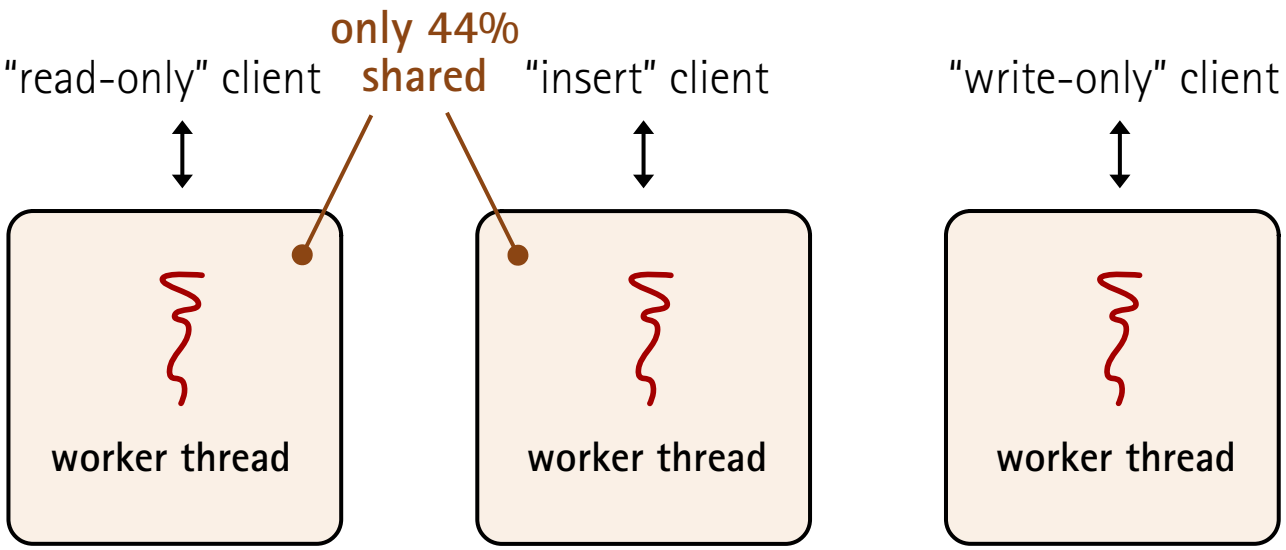
significant code reduction, but still *only* for libraries

→ Poor <u>overall</u> attack-surface reduction



Code Usage (Function Granularity)

MariaDB: **52%** of the code in bytes (≙ **60%** of the functions) stems from the **main binary**



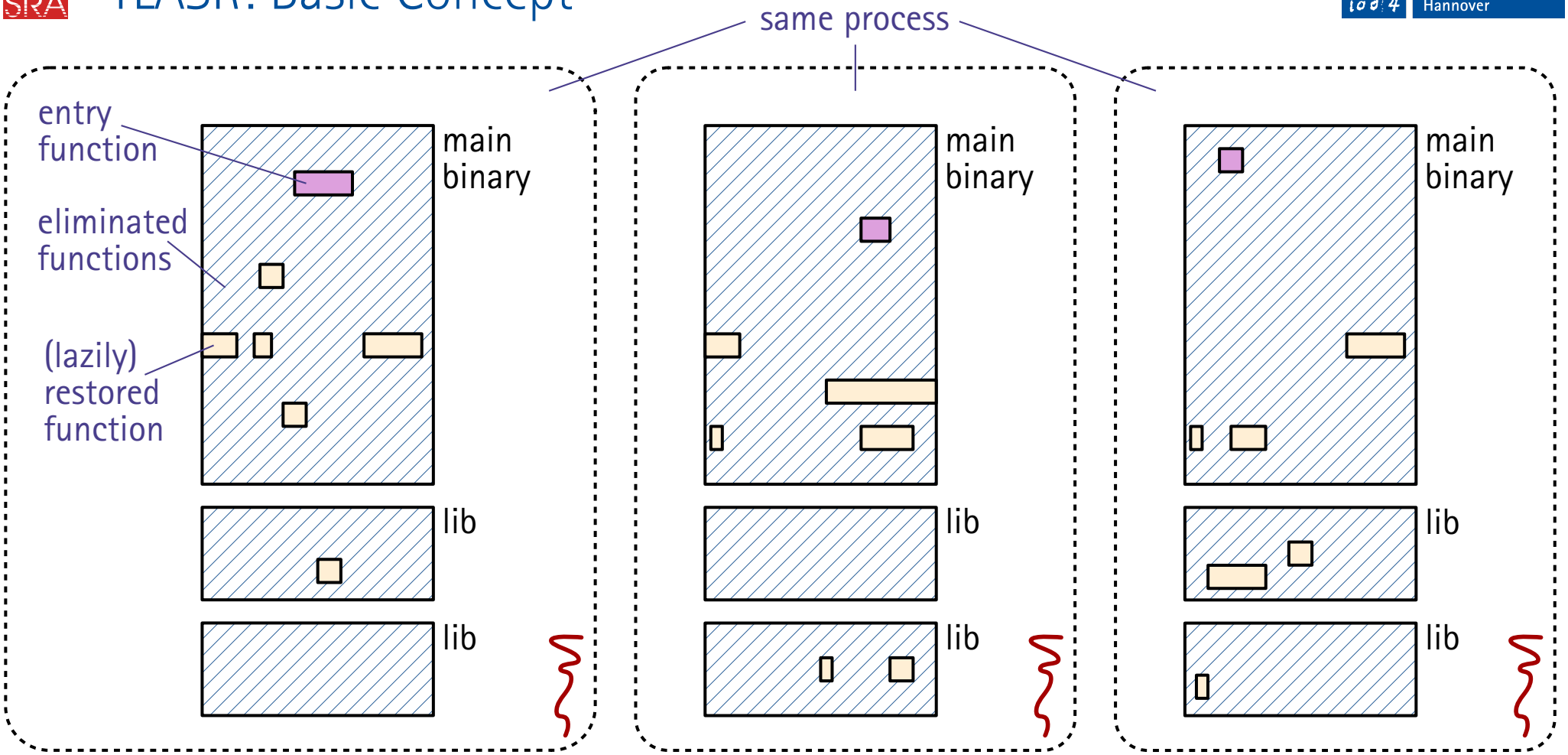
...

Consider
- main binary
- thread contexts

Thread-Level Attack-Surface Reduction (TLASR)

- **Dynamic** – Eliminates functions during runtime / on-demand restoration
  Enhanced by static call-graph analysis
- **Whole process** – Considers the main executable and libraries
- **Per thread** – Works on the context of individual threads

TLASR: Basic Concept

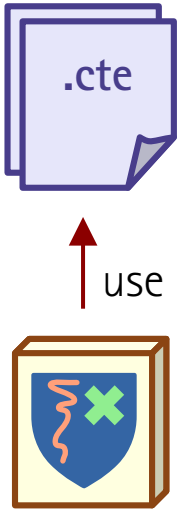


CTE: Eliminating and Restoring Functions

CTE: context-based .text elimination

→ *Binary analysis tool (CTEmeta) + runtime library (libCTE)*

- **CTEmeta** - *ahead of time*
  - Gathers static callgraph information
  - Requires only ELF symbols (no source code, no debug information)
- **libCTE** - *runtime*
  - Eliminates functions at runtime, restores them on call
  - Uses call-graph info (colleted by CTEmeta) to validate function restores
  - (Currently) manual integration into the program



CTE: Killing, Wiping, and Restoring Functions

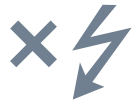
on call

```
function1:
  lea   eax,[rsi+rsi*8]
  add   eax,edi
  ret
```

kill

```
function1:
  int3
  int3
  int3
```

ROP attack



dynamic restoration:

```
function2:
  sub   rsp,0x8
  mov   edx,0x30
  mov   edi,0x1
  xor   eax,eax
  lea   rsi,[rip+0xca3]
  ...
```

wipe

```
function2:
  mov   rax, restore
  call  rax
  .word <fn index>
  int3
  int3
  ...
```

implant

restore

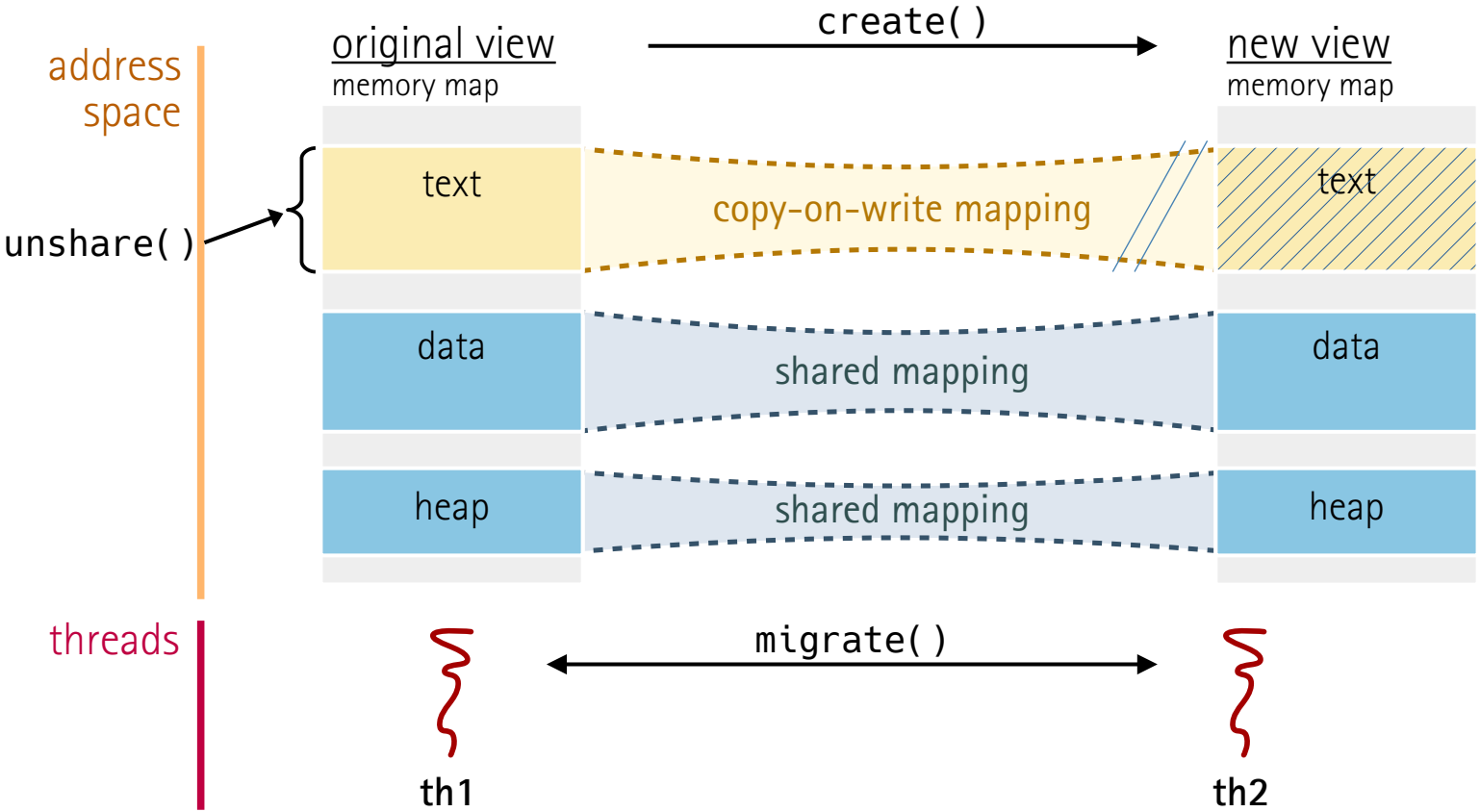
Validation
→ call site
→ call graph
(full CFI)

```
function2:
  sub   rsp,0x8
  mov   edx,0x30
  mov   edi,0x1
  xor   eax,eax
  lea   rsi,[rip+0xca3]
  ...
```

static CTEmeta-generated info

Adress-Space Views (for Linux) (Rommel et al. '20)

Kernel extension: Multiple synchronized address-space clones per process



MariaDB: Per-Worker Views

```
int main(...) {
      ...
+     cte_init();
+     cte_view_unshare();
      ...
}
```

```
mysql_thread_create(...)
```

```
void do_handle_one_connection(...) {
      // mariadb thread/connection init
      ...

+     long thread_view = view_create();
+     view_migrate(thread_view);

+     cte_rules *R = cte_rules_init(CTE_WIPE);
+     cte_wipe(R);

      while (thd_is_connection_alive(thd)) {
            do_command(thd);
      }

                                  adb thread/connection cleanup
```

create view & migrate

configure & perform wipe

```
// load/kill/wipe (→policy) individual functions
cte_rules_set_func(R, policy, func_addr, recursive_callgraph);
cte_rules_set_fnmatch(R, policy, fnmatch_pattern, recursive_callgraph);
cte_rules_set_indirect(R, policy);
```

Attack-Surface Reduction

- **MariaDB** *(client: all sysbench SQL benchmarks)*

  Original: **55614** functions (16.91 MiB)

TLASR:	min:	**1822** functions (1.09 MiB)	**-97%** (-94%)	- *oltp_point_select*
	max:	**2929** functions (1.65 MiB)	**-95%** (-90%)	- *oltp_read_write*

  wiping per connection

- **memcached** *(client: memtier benchmark)*

  Original: **5562** functions (2063 KiB)

TLASR:	min:	**111** functions (46KiB)	**-98%** (-98%)	- *slabs thread*
	max:	**264** functions (115 KiB)	**-95%** (-94%)	- *worker thread*

  wiping per thread

Attack-Surface Reduction

Is an auto-ROP generator[1] able to conduct ROP chain attacks?

- **MariaDB**

  Original: yes

  Static Debloating[2]: yes (-33% of functions)

  TLASR: no (-95% to -97% of functions)

- **memcached**

  Original: yes

  Static Debloating[2]: yes (-65% of functions)

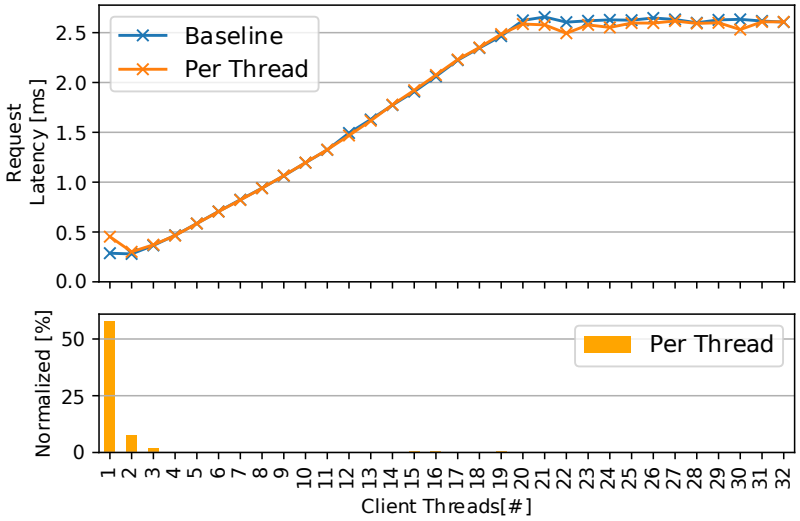
  TLASR: no (-95% to -98% of functions)

[1] ROPgadget tool: http://shell-storm.org/project/ROPgadget

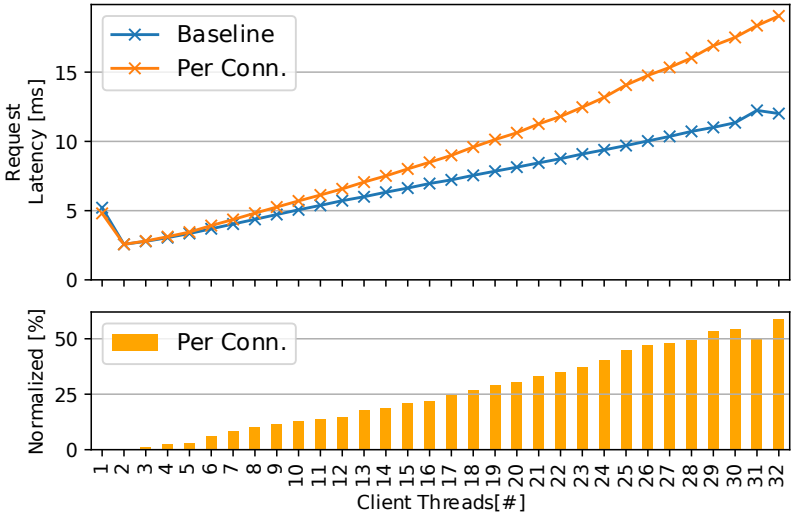
[2] Ziegler et al. '19

Performance Overhead

memcached (thread pool, CPU count)

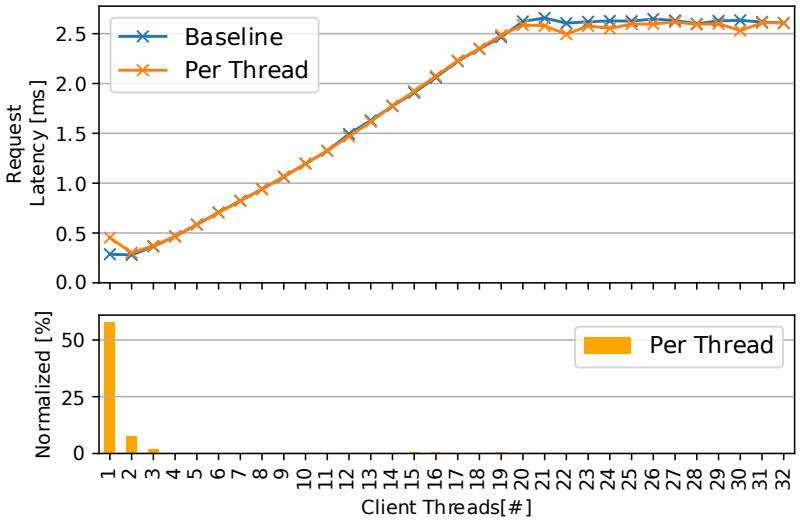


MariaDB (thread per connection)



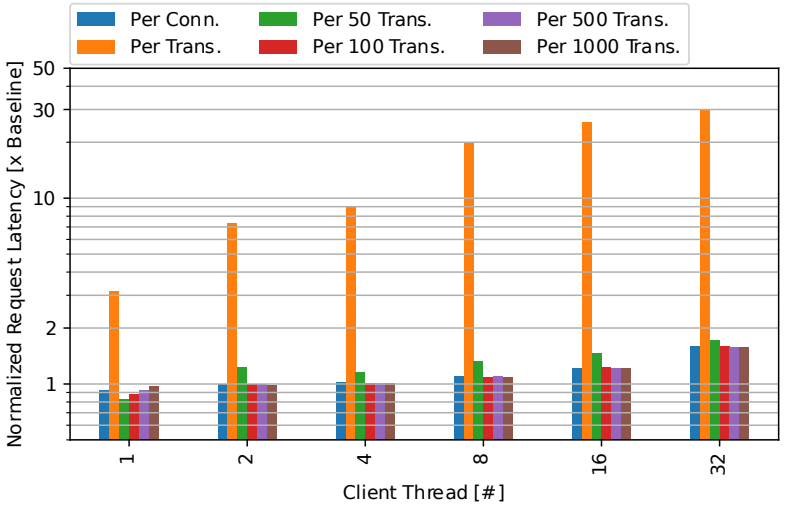
Performance Overhead

memcached (thread pool, CPU count)



MariaDB (thread per connection)

re-wipe between transactions



Summary

Thread-Level Attack-Surface Reduction (TLASR)

- Goal: Reduce attack surface of processes
- Approach: Context-based elimination of code
  - *On-demand* function elimination & restoration of the *whole process*
  - Works on *thread-level* via *address-space views* in Linux
- Results:
  - Reduced attack surface up to -98%
  - Auto-ROP utility turned ineffective with TLASR
  - Reasonable overhead