DISTRIBUTED OPEN RELIABLE SYSTEMS ANALYSIS LAB (DORSAL)

A Novel Approach To Dynamic Instrumentation by exploiting compiler padding

Balboul Anas

Pr. Michel Dagenais

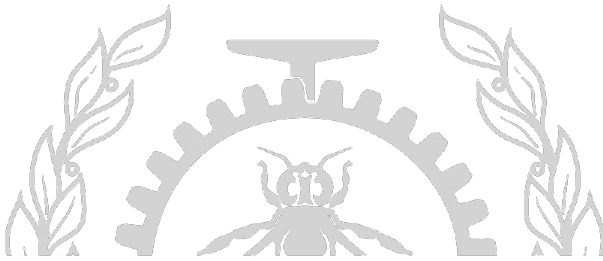
Dec 09, 2019

Polytechnique Montréal

Génie informatique et logiciel

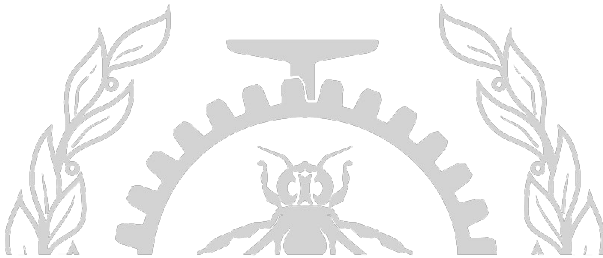
Sommaire

1. Problem definition
2. Proposed solution
3. Limitations
4. Conclusion



1. Problem definition

- Dynamic instrumentation of function entry and exit.
- Highly successful dynamic instrumentation.
- Fast dynamic tracepoints.
- Effective memory usage.



1. Problem definition

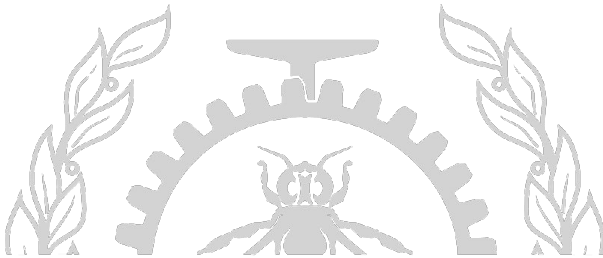
- Instrumentation is problematic when it overlaps more than one instruction:

```
e9 37 ff ff ff        jmpq   91763 <trace_entry>
```

```
0000000000084990 <ngx_http_fastcgi_create_key>:
   84990:       53                      push   %rbx
   84991:       48 8b 47 40             mov    0x40(%rdi),%rax
   84995:       48 89 fb                mov    %rdi,%rbx
   84998:       48 8d b8 c8 00 00 00    lea    0xc8(%rax),%rdi
   8499f:       e8 cc 65 f9 ff          callq  1af70 <ngx_array_push>
   849a4:       48 85 c0                test   %rax,%rax
   849a7:       74 37                   je     849e0 <ngx_http_fastcgi_create_key+0x50>
   849a9:       48 8b 53 28             mov    0x28(%rbx),%rdx
   849ad:       48 8b 0d cc 9c 23 00    mov    0x239ccc(%rip),%rcx        # 2be680 <ngx_
   849b4:       48 89 df                mov    %rbx,%rdi
   849b7:       48 8b 34 ca             mov    (%rdx,%rcx,8),%rsi
   849bb:       48 89 c2                mov    %rax,%rdx
   849be:       48 81 c6 50 02 00 00    add    $0x250,%rsi
   849c5:       e8 26 51 fd ff          callq  59af0 <ngx_http_complex_value>
   849ca:       48 85 c0                test   %rax,%rax
   849cd:       0f 95 c0                setne  %al
   849d0:       0f b6 c0                movzbl %al,%eax
   849d3:       48 f7 d8                neg    %rax
   849d6:       5b                      pop    %rbx
   849d7:       c3                      retq
   849d8:       0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
   849df:       00
   849e0:       48 c7 c0 ff ff ff ff    mov    $0xffffffffffffffff,%rax
   849e7:       5b                      pop    %rbx
   849e8:       c3                      retq
   849e9:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)
```

2. Proposed solution

- To minimize the instrumentation overlapping instructions case:
  - We use a smaller instrumentation instructions.
  - Relative jump 2 bytes has lesser chance than a relative jump 5 bytes.
- But how can we reach the function that does the tracing with only a one byte displacement ?!



2. Proposed solution

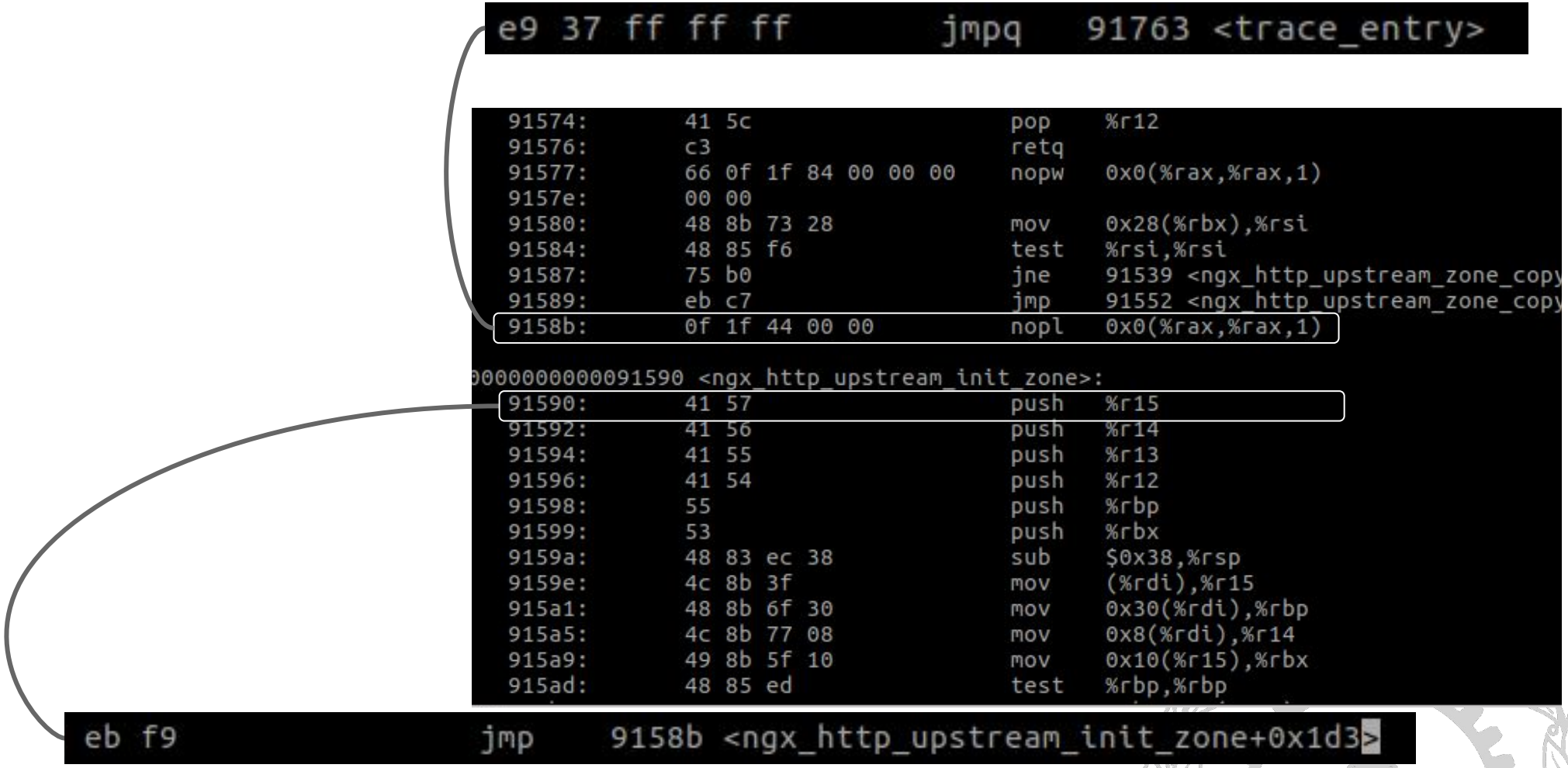
- Binaries built with the following options contain NOP instructions as padding for performance reasons:
  - -falign-functions
  - -falign-jumps
  - -falign-labels
  - -falign-loops
- -O2 and more
  - turns on all these option.

```
  91574:       41 5c                   pop    %r12
  91576:       c3                      retq
  91577:       66 0f 1f 84 00 00 00    nopw   0x0(%rax,%rax,1)
  9157e:       00 00
  91580:       48 8b 73 28             mov    0x28(%rbx),%rsi
  91584:       48 85 f6                test   %rsi,%rsi
  91587:       75 b0                   jne    91539 <ngx_http_upstream_zone_copy
  91589:       eb c7                   jmp    91552 <ngx_http_upstream_zone_copy
  9158b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000091590 <ngx_http_upstream_init_zone>:
  91590:       41 57                   push   %r15
  91592:       41 56                   push   %r14
  91594:       41 55                   push   %r13
  91596:       41 54                   push   %r12
  91598:       55                      push   %rbp
  91599:       53                      push   %rbx
  9159a:       48 83 ec 38             sub    $0x38,%rsp
  9159e:       4c 8b 3f                mov    (%rdi),%r15
  915a1:       48 8b 6f 30             mov    0x30(%rdi),%rbp
  915a5:       4c 8b 77 08             mov    0x8(%rdi),%r14
  915a9:       49 8b 5f 10             mov    0x10(%r15),%rbx
  915ad:       48 85 ed                test   %rbp,%rbp
```

2. Proposed solution

- NOPs after the previous function epilogue can be used (safetly) to insert a trampoline.

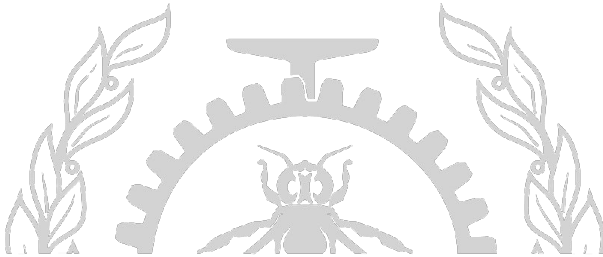


2. Proposed solution

- What if we don’t find a suitable NOP in the end of a function ?
- No problem ! NOPs used for padding loops, labels and jumps may be used as well.

```
   42186:       48 89 d8                mov    %rbx,%rax
   42189:       5b                      pop    %rbx
   4218a:       5d                      pop    %rbp
   4218b:       c3                      retq
   4218c:       0f 1f 40 00             nopl   0x0(%rax)

0000000000042190 <ngx_regex_malloc>:
   42190:       48 89 fe                mov    %rdi,%rsi
   42193:       48 8b 3d be 31 29 00    mov    0x2931be(%rip),%rdi        # 2d5358 <ngx_pcre_pool>
   4219a:       48 85 ff                test   %rdi,%rdi
   4219d:       74 09                   je     421a8 <ngx_regex_malloc+0x18>
   4219f:       e9 1c 8a fd ff          jmpq   1abc0 <ngx_palloc>
   421a4:       0f 1f 40 00             nopl   0x0(%rax)
   421a8:       31 c0                   xor    %eax,%eax
   421aa:       c3                      retq
```



2. Proposed solution

- All NOPs are not safe to use:

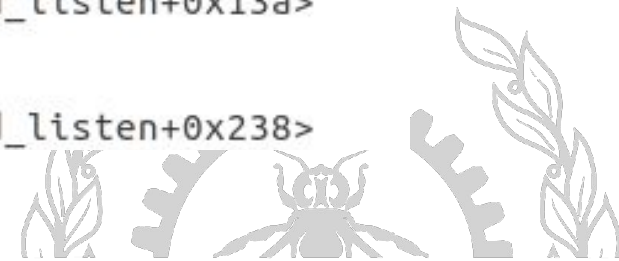
```
 432af:       31 c0                   xor    %eax,%eax
 432b1:       e8 4a 69 fd ff          callq  19c00 <ngx_log_error_core>
 432b6:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 432bd:       00 00 00
 432c0:       49 c7 c5 ff ff ff ff    mov    $0xffffffffffffffff,%r13
 42548:       49 39 6d 08             cmp    %rbp,0x8(%r13)
 4254c:       0f 87 6f ff ff ff       ja     424c1 <ngx_regex_exec_array+0x21>
 42552:       66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
 42558:       48 c7 c5 fb ff ff ff    mov    $0xfffffffffffffffb,%rbp
 4255f:       eb 91                   jmp    424f2 <ngx_regex_exec_array+0x52>
 42561:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 42568:       00 00 00
 4256b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000042570 <ngx_http_merge_locations>:
 42570:       48 85 f6                test   %rsi,%rsi
 42573:       0f 84 ef 00 00 00       je     42668 <ngx_http_merge_locations+0xf8>
 42579:       41 57                   push   %r15
 4257b:       41 56                   push   %r14
 43312:       4c 89 ac 24 b8 00 00    mov    %r13,0xb8(%rsp)
 43319:       00
 4331a:       66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
 43320:       48 81 78 48 48 54 54    cmpq   $0x50545448,0x48(%rax)
 43327:       50
 43328:       0f 85 ad 01 00 00       jne    434db <ngx_http_block+0x59b>
```

2. Proposed solution

- Unreachable NOPs are safe to use:
  - If there is an unconditional branch instruction before it.
  - If there is **no** branching instruction in the function with the address of the NOP as a target.

```
4472a:       eb 9d                   jmp    446c9 <ngx_http_add_listen+0x99>
4472c:       0f 1f 40 00             nopl   0x0(%rax)
44730:       48 8b 8d 40 01 00 00    mov    0x140(%rbp),%rcx
44737:       48 85 c9                test   %rcx,%rcx
4473a:       0f 84 50 01 00 00       je     44890 <ngx_http_add_listen+0x260>
44740:       48 8b b5 48 01 00 00    mov    0x148(%rbp),%rsi
44747:       48 85 f6                test   %rsi,%rsi
4474a:       74 27                   je     44773 <ngx_http_add_listen+0x143>
4474c:       4c 3b 21                cmp    (%rcx),%r12
4474f:       0f 84 13 01 00 00       je     44868 <ngx_http_add_listen+0x238>
44755:       31 c0                   xor    %eax,%eax
44757:       eb 11                   jmp    4476a <ngx_http_add_listen+0x13a>
44759:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)
44760:       4c 3b 24 c1             cmp    (%rcx,%rax,8),%r12
44764:       0f 84 fe 00 00 00       je     44868 <ngx_http_add_listen+0x238>
```

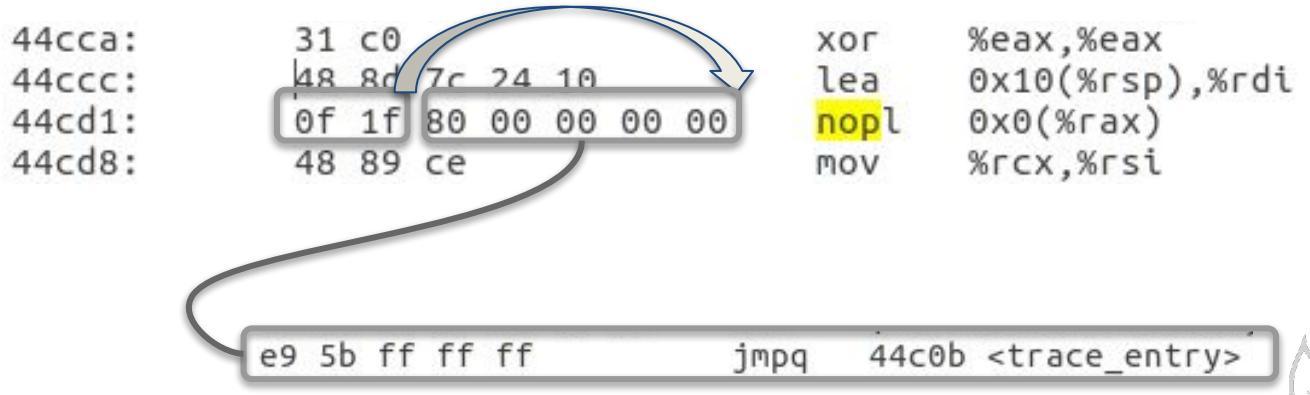


2. Proposed solution

- What if the NOP is reachable ?

```
432af:      31 c0                   xor    %eax,%eax
432b1:      e8 4a 69 fd ff          callq  19c00 <ngx_log_error_core>
432b6:      66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
432bd:      00 00 00
432c0:      49 c7 c5 ff ff ff ff    mov    $0xffffffffffffffff,%r13
```

- We can still use it if its size is superior to 7:
  - By using a 2 bytes jump to jump over the nop and use the rest as a trampoline:



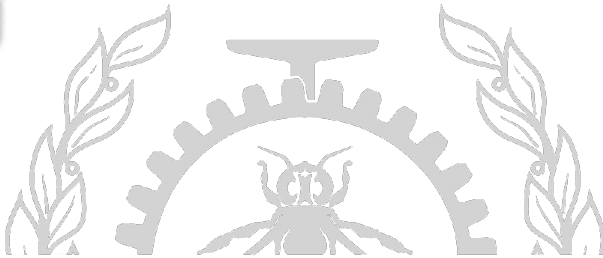
2. Proposed solution

- We can still use it if its size is superior to 7:
  - By changing the operand and leaving the opcode untouched, thus embedding the trampoline in the operand:

```
432af:      31 c0                   xor    %eax,%eax
432b1:      e8 4a 69 fd ff          callq  19c00 <ngx_log_error_core>
432b6:      66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
432bd:      00 00 00
432c0:      49 c7 c5 ff ff ff ff    mov    $0xffffffffffffffff,%r13
```

```
432af:      31 c0                   xor    %eax,%eax
432b1:      e8 4a 69 fd ff          callq  19c00 <ngx_log_error_core>
432b6:      66 2e 0f 1f 84 e9 5b    nopw   %cs:0x8(%rcx,%rbp,0xa5)
432bd:      ff ff ff
432c0:      49 c7 c5 ff ff ff ff    mov    $0xffffffffffffffff,%r13
```

```
e9 5b ff ff ff          jmpq   44c0b <trace_entry>
```



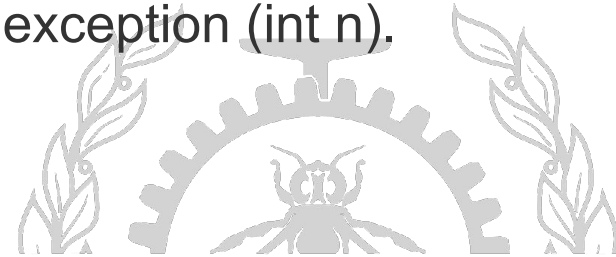
2. Proposed solution

- What if we don't find any NOP near the instrumented point ?
- We use a relative jump 32 as an instrumentation instruction instead of relative jump 8:

```
e9 5b ff ff ff          jmpq   44c0b <trace_entry>
```

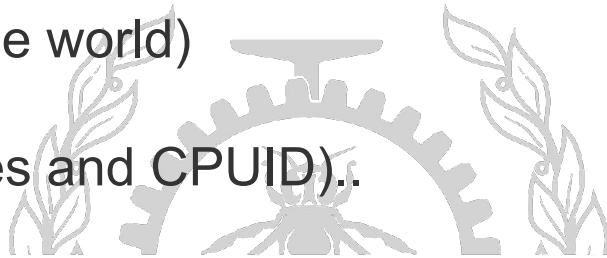
```
0000000000042da0 <ngx_http_init_static_location_trees.isra.7>:
   42da0:       31 c0                   xor    %eax,%eax
   42da2:       48 85 d2                test   %rdx,%rdx
   42da5:       0f 84 5d 01 00 00       je     42f08 <ngx_http_init_static_location_trees.isra.7+0x168>
   42dab:       4c 8b 02                mov    (%rdx),%r8
   42dae:       49 39 d0                cmp    %rdx,%r8
   42db1:       0f 84 51 01 00 00       je     42f08 <ngx_http_init_static_location_trees.isra.7+0x168>
   42db7:       41 56                   push   %r14
```

- There should be no branching to the patched area.
- No indirect branching in the function.
- Function contains no instruction that causes an exception (int n).



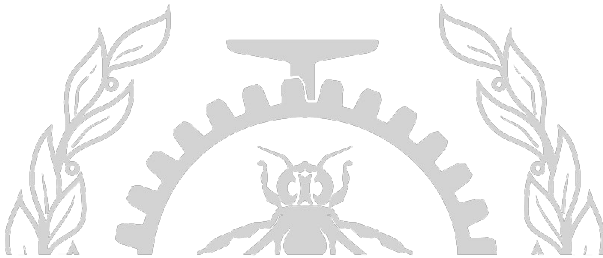
2. Proposed solution

- For dynamic tracing, instrumentation should be done on the fly:
  - The jump 2 bytes used for the entry function is done atomically.
  - If the NOP is unreachable which is the case 95% of the time, we just patch it without worrying about multi-thread problems.
  - If it's reachable and has a size superior to 7, we can embed a trampoline in it without worrying about corrupting the NOP. If the size is inferior to 8 we do it atomicly (in x64 at least).
  - Needs addressing some simultaneous execution issues on preemptive kernel.. (preferably without stopping the world)
  - And cross self-modifying issues (cache boundaries and CPUID)..



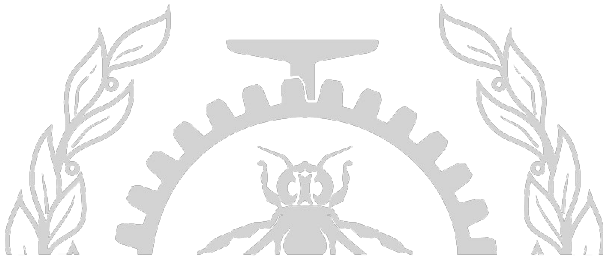
2. Proposed solution

- Results:
  - Success rate:
    - uftrace: 86%
    - nginx: 89%
    - Git: 82%
    - V8 JavaScript engine (Google chrome): 90.46%



3. Limitations and Problems

- Limited to function entry and exit.
- If the binary is not optimized, NOPs will be missing.
- A greedy algorithm is used to find a suitable NOP. An optimal algorithm may be used to increase the success rate even more.



Demonstration 1

Questions?

anasbalbo@gmail.com

https://github.com/AnsBal/