On Building the Data-Oblivious Virtual Environment

Tushar M. Jois, Hyun Bin Lee, Christopher W. Fletcher, Carl A. Gunter

Learning from Authoritative Security Experiment Results (LASER) Workshop
February 25, 2021
Function $f$

Cloud SGX Enclave

Prevents the OS from introspection on or tampering with the computation.

Input $i_1$: 01000101 01000101

Input $i_2$: 10111010 10111010

UIUC: Usually written in a high-level language, like R.

Johns Hopkins: microarchitectural side-channel attack

Result $r$: 1101

Outsourcing scientific computation

Infer secrets of computation by reading states of the microarchitecture.
data-oblivious computation (n.)

a program execution with the same observable characteristics regardless of the inputs provided

see also constant-time programming
Key insight:

data-oblivious →
data-unnecessary
Separate the execution of a script from the operations on sensitive data.

**Function f**

Input $i_1 = 01000101$

Input $i_1 = 01000101$

Result $r_1 = 000110100100$

Result $r_1 = 000110100100$

**Data Oblivious Transcript**

**Execution Trace on a Dataset**
DOVE: A Data-Oblivious Virtual Environment
Hyun Bin Lee, Tushar M. Jois, Christopher W. Fletcher, Carl A. Gunter
NDSS 2021
Our goal:

Design the first data-oblivious R stack.
A soft selective sweep during rapid evolution of gentle behaviour in an Africanized honeybee

Arian Avalos, Hailin Pan, Cai Li, et al.

Nature Communications 2017 8(1) 1550
Why use the bee study?

- A real, publicly-available dataset (1.3 GB)
- Similar to human genomics workloads
- Cross-university collaboration
- R code from a repository of genomics scripts
A case study for evaluating the data-obliviousness of R
Experimentally evaluating data-obliviousness

```
calc_snp_stats ← function(geno)
```
Two types of problematic instructions:

- Variable-time instructions
- Conditional jumps on sensitive data

Instructions from `libfixedtimefixedpoint`

```
add  mov  pop  setg
and  movabs  push  setl
call  movsd  rep  settle
cdqe  movsx  ret  setne
cmp  movsxid  sar  shli
mul  movzx  sbb  shr
je
jmp
jne
lea
mul
neg
not
or
seta
setae
setbe
sete
setne
setnp
setle
setlne
setngp
setno
setnp
setne
setpl
setp
setpe
setpl
setz
test
xor
```

On Subnormal Floating Point and Abnormal Timing
Marc Andrysco, David Kohlbrenner, Keaton Mowery, et al.
IEEE S&P (Oakland) 2015

Conditional jumps must **NOT** touch sensitive data
Instruction count

(gdb) break Enclave/runtime.cpp:327
(gdb) commands 1
Type commands for breakpoint(s) 1, one per line. End with a line saying just "end".
> set record btrace bts buffer-size unlimited
> record btrace
> continue
> end
(gdb) run

Recorded 1278564 instructions in 84466 functions (0 gaps)
 Intel Performance Counter Monitor (PCM)

- **cycle counts**
  - getCycles
  - getCyclesLostDueL3CacheMisses
  - getCyclesLostDueL2CacheMisses

- **bytes to/from memory controller**
  - getBytesReadFromMC
  - getBytesWrittenToMC
  - getIOPrequestBytesFromMC

- **cache hits & misses**
  - getL2CacheHitRatio
  - getL3CacheHitRatio
  - getL3CacheMisses
  - getL2CacheMisses
  - getL2CacheHits
  - getL3CacheHitsNoSnoop
  - getL3CacheHitsSnoop
  - getL3CacheHits
calc_snp_stats <- function(geno) {
  ## Eva KF Chan
  ## http://evachan.org

  m <- nrow(geno)  ## number of snps
  n <- ncol(geno)  ## number of individuals

  geno[[geno != 0] & (geno != 1) & (geno != 2)] <- NA

  geno <- as.matrix(geno)

  n0 <- apply(geno == 0, 1, sum, na.rm = T)
  n1 <- apply(geno == 1, 1, sum, na.rm = T)
  n2 <- apply(geno == 2, 1, sum, na.rm = T)
  n <- n0 + n1 + n2

  ## (snip) ##
}
Instructions in compiled binary

Conditional branches on data

Intel PCM

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Instr. Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 &amp; 0</td>
<td>0</td>
<td>45</td>
</tr>
<tr>
<td>0 &amp; 1</td>
<td>0</td>
<td>45</td>
</tr>
<tr>
<td>0 &amp; NA</td>
<td>0</td>
<td>45</td>
</tr>
<tr>
<td>1 &amp; 0</td>
<td>0</td>
<td>47</td>
</tr>
<tr>
<td>NA &amp; 0</td>
<td>0</td>
<td>47</td>
</tr>
<tr>
<td>NA &amp; 1</td>
<td>NA</td>
<td>53</td>
</tr>
<tr>
<td>NA &amp; NA</td>
<td>NA</td>
<td>53</td>
</tr>
<tr>
<td>1 &amp; 1</td>
<td>1</td>
<td>54</td>
</tr>
<tr>
<td>1 &amp; NA</td>
<td>NA</td>
<td>57</td>
</tr>
</tbody>
</table>
geno[(geno ≠ 0) & (geno ≠ 1) & (geno ≠ 2)] ← NA

# R interpreter implementation of &
if (x1 == 0 || x2 == 0)
  pa[i] = 0;
else if (x1 == NA || x2 == NA)
  pa[i] = NA;
else
  pa[i] = 1;
}
R interpreter

- **Fortran**
  - 258,876 SLOC (26.1%)

- **R**
  - 345,547 SLOC (34.8%)

- **C**
  - 388,141 SLOC (39.1%)
## Solution design

**Build a data-oblivious virtual environment**

Correctness

Data-obliviousness

- Instructions in compiled binary
- Instruction count
- Intel PCM

Expressiveness

Efficiency
Solution design

Build a data-oblivious virtual environment

Correctness

Data-obliviousness

Instructions in compiled binary

Instruction count

Intel PCM

Expressiveness

Efficiency
**Local Machine**

R Script $f$

**Input $i_2$**


---

**Cloud SGX Enclave**

Data-Oblivious Virtual Environment

Load DOT and Datasets

Run DOT on Datasets

Result $r$

1101

---

**Input $i_1$**

01000101
01000101

**Input $i_2$**

10111010
10111010
ecall_dispatch();

instr* t = parser.get_next();
p_block* result = alloc_result_matrix(t);
line_dispatch(t, result);

line_dispatch(instr* t, p_block* result);

vector<p_block*> args = t->args();
Op* operation = op_factory(t->name);
operation->call(args[0], args[1], result);

AddOp::call(p_block* A, B, C);

for (i, j in 0:C→nrow, 1:C→ncol)
    call(A[i,j], B[i,j], C[i,j]);

AddOp::call(fixed* A_ij, B_ij, C_ij);
*C_ij = fix_add(*A_ij, *B_ij);
Operation on scalars

```
ecall_dispatch();

instr* t = parser.get_next();
p_block* result = alloc_result_matrix(t);
line_dispatch(t, result);

line_dispatch(instr* t, p_block* result);

vector<p_block*> args = t->args();
Op* operation = op_factory(t->name);
operation->call(args[0], args[1], result);

AddOp::call(p_block* A, B, C);
for (i, j in 0:C->nrow, 1:C->ncol)
    call(A[i,j], B[i,j], C[i,j]);

AddOp::call(fixed* A_iJ, B_iJ, C_iJ);
*C_iJ = fix_add(*A_iJ, *B_iJ);
```

Argument loading

Iteration over data pointers in matrix

Leaf Function

Data-obliviousness should be tested here
Side-channels in leaf functions

Instructions in compiled binary

\texttt{cmovne}

Instruction count

\textbf{TESTS}
- Testing Abs (1/45)...
- Testing Abs, ratio 0.1...
- Testing Abs, ratio 0.2...

Passed

\textbf{Intel PCM}

![Box plot showing cycle counts for different proportions of genotype 0s.](image)
\text{geno}[(\text{geno} \neq 0) \land (\text{geno} \neq 1) \land (\text{geno} \neq 2)] \rightarrow \text{NA}
Solution design

Build a data-oblivious virtual environment

Correctness

Data-obliviousness

Instructions in compiled binary

Instruction count

Intel PCM

Expressiveness

Efficiency
abs  sqrt  floor  ceiling  exp  log  cos  sin  tan  sign  +  -  *  /  ^  %%  %/%  >  <  ?=  ?=  ?=  |  &  !  all  any  sum  prod  min  max  range  is.na  is.nan  is.infinite  if  %*%  cbind  rbind  for  matrix  dim
Solution design

Build a data-oblivious virtual environment

Correctness

Data-obliviousness

Instructions in compiled binary

Instruction count

Intel PCM

Expressiveness

Efficiency
**O(m * n) space**

2,808,570×60 dataset

(*)

**O(m^2) space**

10,000×60 dataset
Solution design

Build a data-oblivious virtual environment

Correctness

Data-obliviousness

Instructions in compiled binary

Instruction count

Intel PCM

Expressiveness

Efficiency

https://github.com/dove-project/benchmarks
Discussion

- Did you use experimentation artifacts borrowed from the community?
- Did you attempt to replicate or reproduce results of earlier research as part of your work?
- What can be learned from your methodology and your experience using your methodology?
- What did you try that did not succeed before getting to the results you
  obtained?
- Did you produce any intermediate results including possible unsuccessful tests or experiments?
- Did you share experimentation artifacts with the community?
Discussion

- Did you use experimentation artifacts borrowed from the community?
- Did you attempt to replicate or reproduce results of earlier research as part of your work?
- What can be learned from your methodology and your experience using your methodology?
- What did you try that did not succeed before getting to the results you did?
- Did you produce any intermediate results including possible unsuccessful tests or experiments?
- Did you share experimentation artifacts with the community?
Discussion

- Did you use experimentation artifacts borrowed from the community?
- Did you attempt to replicate or reproduce results of earlier research as part of your work?
- What can be learned from your methodology and your experience using your methodology?
- What did you try that did not succeed before getting to the results you
- Did you produce any intermediate results including possible unsuccessful tests or experiments?
- Did you share experimentation artifacts with the community?
Intermediate results: data-obliviousness

- Fisher test is used in script to measure deviation from Hardy-Weinberg Equilibrium
- Originally a part of external library, didn’t test it, but clearly wrong assumption
  - When we started to look at it, saw it failed our instruction tests -- factorials
  - Rewrote it to use front-end primitives -- worse performance, but security guaranteed (and smaller TCB)
- Insecure (4.9x overhead) to secure (315x overhead)

\[
p = \frac{\binom{a+b}{a} \binom{c+d}{c}}{\binom{n}{a+c}} = \frac{(a+b)! \ (c+d)! \ (a+c)! \ (b+d)!}{a! \ b! \ c! \ d! \ n!}
\]
Intermediate results: expressiveness

- Original DOVE design required end-users to modify their R code before a DOT was generated
- Not a good design
  - restricts expressiveness to what the user knows how to write using DOVE
  - Might as well learn a new language
- Created an automator that instruments R base functions & structures to use DOVE counterparts
  - No need to manually write DOVE

```
# Original version (works in current DOVE)
gen0[(geno!=0) & (geno!=1) & (geno!=2)] ← NA
geno ← as.matrix(geno)
n0 ← apply(geno==0,1,sum,na.rm=T)
n1 ← apply(geno==1,1,sum,na.rm=T)
n2 ← apply(geno==2,1,sum,na.rm=T)

# Pre-automation version
geno ← +geno
gen0[(geno!=C_0) & (geno!=C_1) & (geno!=C_2)] ← NA
n0 ← rowSums(geno==C_0,na.rm=T)
n1 ← rowSums(geno==C_1,na.rm=T)
n2 ← rowSums(geno==C_2,na.rm=T)
```
Intermediate results: efficiency

- Originally didn’t have for loops
  - Applications used apply, rowSums, and similar
- Applications that used loops had awful performance
  - Loops would just get unrolled
  - DOT became size $O(n)$
- Performance made us realize that loops were important enough
  - apply wasn’t enough
  - So, we implemented it

<table>
<thead>
<tr>
<th>Script</th>
<th>Overhead before for</th>
<th>Overhead after for</th>
</tr>
</thead>
<tbody>
<tr>
<td>allele_sharing</td>
<td>295x</td>
<td>105x</td>
</tr>
<tr>
<td>EHHS*</td>
<td>1246x</td>
<td>189x</td>
</tr>
<tr>
<td>iES*</td>
<td>1204x</td>
<td>154x</td>
</tr>
<tr>
<td>LD*</td>
<td>220x</td>
<td>18x</td>
</tr>
</tbody>
</table>
Discussion

- Did you use experimentation artifacts borrowed from the community?
- Did you attempt to replicate or reproduce results of earlier research as part of your work?
- What can be learned from your methodology and your experience using your methodology?
- What did you try that did not succeed before getting to the results you
- Did you produce any intermediate results including possible unsuccessful tests or experiments?
- Did you share experimentation artifacts with the community?

https://github.com/dove-project/benchmarks
benchmarks

Introduction

This repository provides scripts to run benchmarks that we used for evaluating DOVE, as well as our benchmark results. You can read more about this in our academic research paper, DOVE: A Data-Oblivious Virtual Environment, which appeared in NDSS 2021.

This is research code, and has not been certified for production use. That being said, if you see something, say something!

Running the benchmarks
Future work

Future plans

● Extend DOVE to more languages and frameworks
● Implement data-oblivious performance enhancement
● Understand what data-oblivious hardware instructions can support a system like DOVE

Post-workshop paper

● Review systematically the R side channels we discovered
● Re-run all benchmarks using most modern versions of the stack
  ○ New versions of libraries, R interpreter
● Several runs of the same benchmarks
  ○ Variance between benchmarks
● Look into performance on other enclaves, if possible
# μArch Vulnerable
# Assume x1, x2 are private

```c
if (x1 \&\& x2) {
    y = 1;
} else {
    y = 0;
}
```

Execution Trace

```
\[
x1 \neq 1 \\
x2 \neq 1 \\
y = 0 \\
y = 1
\]
```

```
x1 \& x2
y = x1 \& x2
```

# Fixed (under assumptions)

```
y = x1 \& x2;
```
Fig. 13: Absolute runtimes and sizes of the evaluation programs. Programs marked with an * were run on a reduced dataset due to test system limitations. Program iES calls EHHS, so we include the lines of code from EHHS when measuring lines of code for iES. FE are measurements for frontend, NEBE are for measurements with backend without SGX, and EBE are for the backend with SGX. F indicates the use of libFTFP, the data-oblivious floating point arithmetic library that we used on our DOVE implementation. LoC stands for Lines of Code for the original R program whereas DOT size represents the size of the counterpart DOT file in bytes. Finally, the DOT overhead represents the relative overhead of the DOT’s file size relative to the size of the original R program.

<table>
<thead>
<tr>
<th>Program</th>
<th>Vanilla R (s)</th>
<th>FE (s)</th>
<th>NEBE (s)</th>
<th>NEBE w/ F (s)</th>
<th>EBE w/ F (s)</th>
<th>LoC (lines)</th>
<th>DOT size (bytes)</th>
<th>DOT Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>EHHS*</td>
<td>18.9</td>
<td>3.85</td>
<td>1104.43</td>
<td>2131.65</td>
<td>3575.46</td>
<td>40</td>
<td>1538</td>
<td>0.51</td>
</tr>
<tr>
<td>iES*</td>
<td>23.48</td>
<td>6.43</td>
<td>1106.34</td>
<td>2161.95</td>
<td>3625</td>
<td>15 + 40</td>
<td>159853</td>
<td>105.44</td>
</tr>
<tr>
<td>LD*</td>
<td>1787.58</td>
<td>3.64</td>
<td>2869.48</td>
<td>9040</td>
<td>32264</td>
<td>54</td>
<td>5610</td>
<td>0.98</td>
</tr>
<tr>
<td>allele_sharing</td>
<td>283.41</td>
<td>5.6</td>
<td>650.03</td>
<td>1841.28</td>
<td>29733</td>
<td>12</td>
<td>419</td>
<td>0.28</td>
</tr>
<tr>
<td>hwe_chisq</td>
<td>38.48</td>
<td>4.56</td>
<td>113.98</td>
<td>262.23</td>
<td>853.49</td>
<td>21</td>
<td>5295</td>
<td>4.35</td>
</tr>
<tr>
<td>hwe_fisher</td>
<td>690.2</td>
<td>4.98</td>
<td>1414.25</td>
<td>154194</td>
<td>234054</td>
<td>12</td>
<td>10287</td>
<td>3.92</td>
</tr>
<tr>
<td>neiFis_multispop</td>
<td>85.85</td>
<td>16.88</td>
<td>111.82</td>
<td>278.42</td>
<td>1077.44</td>
<td>38</td>
<td>5311</td>
<td>4.09</td>
</tr>
<tr>
<td>neiFis_onepop</td>
<td>39.13</td>
<td>4.9</td>
<td>55.85</td>
<td>192.53</td>
<td>764.38</td>
<td>19</td>
<td>7381</td>
<td>2.43</td>
</tr>
<tr>
<td>snp_stats</td>
<td>692.73</td>
<td>11.21</td>
<td>1427.83</td>
<td>155840</td>
<td>236644</td>
<td>33</td>
<td>1980</td>
<td>1.35</td>
</tr>
<tr>
<td>wcFstats</td>
<td>55.27</td>
<td>8.21</td>
<td>79.38</td>
<td>186.27</td>
<td>757.38</td>
<td>35</td>
<td>6624</td>
<td>1.58</td>
</tr>
<tr>
<td>wcFst_spop_pairs</td>
<td>74.05</td>
<td>15.43</td>
<td>206.55</td>
<td>458.26</td>
<td>1343.51</td>
<td>45</td>
<td>18606</td>
<td>5.21</td>
</tr>
</tbody>
</table>