Project Milestone Report

CHANHEE CHO, HENRY JUNG

MAJOR CHANGES

In our initial proposal, we planned to implement compiler-based transformations to mitigate side-channel attacks based on control-flow and data-flow, along with implementing a heuristic for reducing performance overhead. There are no major changes with these goals; however based on our current progress, we may not reach the 125% goal for implementing heuristics, but are on track with the main implementation component (transformation passes).

WHAT YOU HAVE ACCOMPLISHED SO FAR

Progress Update: We have implemented a LLVM pass for timing side channel mitigations due to control flow, based on [1] and [2]. The pass works at the IR-level in optimization, performing if-conversion on control-flow patterns (if-else diamond and if-then triangle), along with adding safe instructions for certain operations (e.g. division, loads, stores). Initial testing has been done on a series of small microbenchmarks along with larger C programs, verifying that control flow has been replaced with conditional execution in assembly (example transformations performed by the LLVM pass listed in Appendix).

MEETING YOUR MILESTONE

We have roughly met our milestone goal with implementing the control-flow transformations. We have not started implementing the transformations for data-flow, but plan on finishing the implementation within the remaining weeks.

SURPRISES

For testing our control-flow mitigation pass, which performs if-conversion, we compile the resulting LLVM bitcode with -O0, as LLVM runs the -simplify-cfg pass for higher optimization levels and performs if-conversion to some degree. However, we observed that certain transformations were not performed compared to our implementations. Our implementation follows the approach in [1] and [2], and we implemented it only referencing LLVM documentation. Even if there are related available implementations we did not discover, we believe it is likely to be different from our approaches.

REVISED SCHEDULE

4/10-4/14: Work on data-flow transformation implementation

4/17-4/21: Finish data-flow transformation implementation; Start work on implementing heuristic to reduce performance overhead

4/24-4/26: Finish evaluating heuristic with experiments; Write report and prepare poster for presenting

RESOURCES NEEDED

(N/A) We have all the resources needed to complete this project.

REFERENCES

[1] B. Coppens, I. Verbauwhede, K. De Bosschere and B. De Sutter, "Practical Mitigations for Timing-Based Side-Channel Attacks on Modern x86 Processors," 2009 30th IEEE Symposium on Security and Privacy, Oakland, CA, USA, 2009, pp. 45-60, doi: 10.1109/SP.2009.19.

[2] Alexander Jordan, Nikolai Kim, and Andreas Krall. 2013. IR-level versus machine-level if-conversion for predicated architectures. In Proceedings of the 10th Workshop on Optimizations for DSP and Embedded Systems (ODES '13). Association for Computing Machinery, New York, NY, USA, 3–10. https://doi.org/10.1145/2443608.2443611

APPENDIX: TRANSFORMATIONS

```
define dso_local i32 @divide(i32 noundef %a, i32 noundef %b) #0 {
entry:
  %cmp = icmp ne i32 %b, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  %div = sdiv i32 %a, %b
  br label %return

if.else:                                          ; preds = %entry
  br label %return

return:                                           ; preds = %if.else, %if.then
  %retval.0 = phi i32 [ %div, %if.then ], [ %a, %if.else ]
  ret i32 %retval.0
}
```

(a) LLVM IR (before transformation)

```
define dso_local i32 @divide(i32 noundef %a, i32 noundef %b) #0 {
entry:
  %cmp = icmp ne i32 %b, 0
  %0 = select i1 %cmp, i32 %a, i32 1
  %div = sdiv i32 %a, %0
  %1 = select i1 %cmp, i32 %div, i32 %a
  ret i32 %1
}

; Function Attrs: noinline nounwind sspstrong uwtable
define dso_local i32 @main() #0 {
entry:
  %call = call i32 @divide(i32 noundef 30, i32 noundef 0)
  %call1 = call i32 (ptr, ...) @printf(ptr noundef @.str, i32 noundef %call)
  ret i32 0
}
```

(b) LLVM IR (after transformation)

Fig. 1. test_division microbenchmark: LLVM IR transformation

```
0000000000000000 <divide>:
   0:   55                      push   rbp
   1:   48 89 e5                mov    rbp,rsp
   4:   89 7d f8                mov    DWORD PTR [rbp-0x8],edi
   7:   89 75 fc                mov    DWORD PTR [rbp-0x4],esi
   a:   83 fe 00                cmp    esi,0x0
   d:   0f 84 11 00 00 00       je     24 <divide+0x24>
  13:   8b 4d fc                mov    ecx,DWORD PTR [rbp-0x4]
  16:   8b 45 f8                mov    eax,DWORD PTR [rbp-0x8]
  19:   99                      cdq
  1a:   f7 f9                   idiv   ecx
  1c:   89 45 f4                mov    DWORD PTR [rbp-0xc],eax
  1f:   e9 0b 00 00 00          jmp    2f <divide+0x2f>
  24:   8b 45 f8                mov    eax,DWORD PTR [rbp-0x8]
  27:   89 45 f4                mov    DWORD PTR [rbp-0xc],eax
  2a:   e9 00 00 00 00          jmp    2f <divide+0x2f>
  2f:   8b 45 f4                mov    eax,DWORD PTR [rbp-0xc]
  32:   5d                      pop    rbp
  33:   c3                      ret
  34:   66 66 66 2e 0f 1f 84    data16 data16 cs nop WORD PTR [rax+rax*1+0x0]
  3b:   00 00 00 00 00
```

(a) Disassembly (before transformation)

```
0000000000000000 <divide>:
   0:   55                      push   rbp
   1:   48 89 e5                mov    rbp,rsp
   4:   89 f8                   mov    eax,edi
   6:   89 45 fc                mov    DWORD PTR [rbp-0x4],eax
   9:   b9 01 00 00 00          mov    ecx,0x1
   e:   83 fe 00                cmp    esi,0x0
  11:   0f 45 c8                cmovne ecx,eax
  14:   99                      cdq
  15:   f7 f9                   idiv   ecx
  17:   89 c1                   mov    ecx,eax
  19:   8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
  1c:   83 fe 00                cmp    esi,0x0
  1f:   0f 45 c1                cmovne eax,ecx
  22:   5d                      pop    rbp
  23:   c3                      ret
  24:   66 66 66 2e 0f 1f 84    data16 data16 cs nop WORD PTR [rax+rax*1+0x0]
  2b:   00 00 00 00 00
```

(b) Disassembly (after transformation)

Fig. 2. test_division microbenchmark: Disassembly transformation

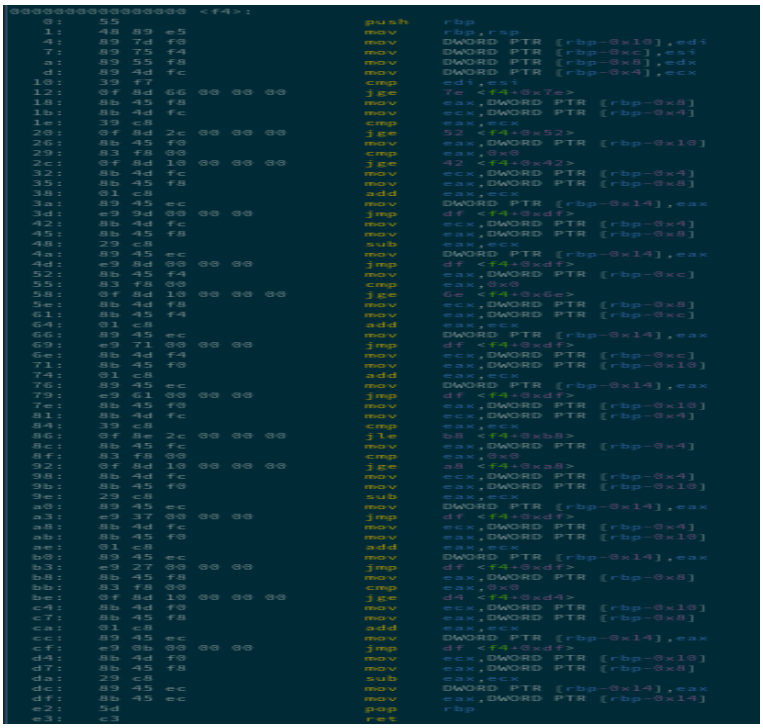
```
define dso_local i32 @f4(i32 noundef %a, i32 noundef %b, i32 noundef %c, i32 noundef %d) #0 {
entry:
  %cmp = icmp slt i32 %a, %b
  br i1 %cmp, label %if.then, label %if.else11

if.then:                                          ; preds = %entry
  %cmp1 = icmp slt i32 %c, %d
  br i1 %cmp1, label %if.then2, label %if.else5

if.then2:                                         ; preds = %if.then
  %cmp3 = icmp slt i32 %a, 0
  br i1 %cmp3, label %if.then4, label %if.else

if.then4:                                         ; preds = %if.then2
  %add = add nsw i32 %c, %d
  br label %return

if.else:                                          ; preds = %if.then2
  %sub = sub nsw i32 %c, %d
  br label %return

if.else5:                                         ; preds = %if.then
  %cmp6 = icmp slt i32 %b, 0
  br i1 %cmp6, label %if.then7, label %if.else9

if.then7:                                         ; preds = %if.else5
  %add8 = add nsw i32 %b, %c
  br label %return

if.else9:                                         ; preds = %if.else5
  %add10 = add nsw i32 %a, %b
  br label %return

if.else11:                                        ; preds = %entry
  %cmp12 = icmp sgt i32 %a, %d
  br i1 %cmp12, label %if.then13, label %if.else19

if.then13:                                        ; preds = %if.else11
  %cmp14 = icmp slt i32 %d, 0
  br i1 %cmp14, label %if.then15, label %if.else17

if.then15:                                        ; preds = %if.then13
  %sub16 = sub nsw i32 %a, %d
  br label %return

if.else17:                                        ; preds = %if.then13
  %add18 = add nsw i32 %a, %d
  br label %return

if.else19:                                        ; preds = %if.else11
  %cmp20 = icmp slt i32 %c, 0
  br i1 %cmp20, label %if.then21, label %if.else23

if.then21:                                        ; preds = %if.else19
  %add22 = add nsw i32 %c, %a
  br label %return

if.else23:                                        ; preds = %if.else19
  %sub24 = sub nsw i32 %c, %a
  br label %return

return:                                           ; preds = %if.else23, %if.then21, %if.else1
  %retval.0 = phi i32 [ %add, %if.then4 ], [ %sub, %if.else ], [ %add8, %if.then7 ], [ %add10
  ret i32 %retval.0
}
```

(a) LLVM IR (before transformation)

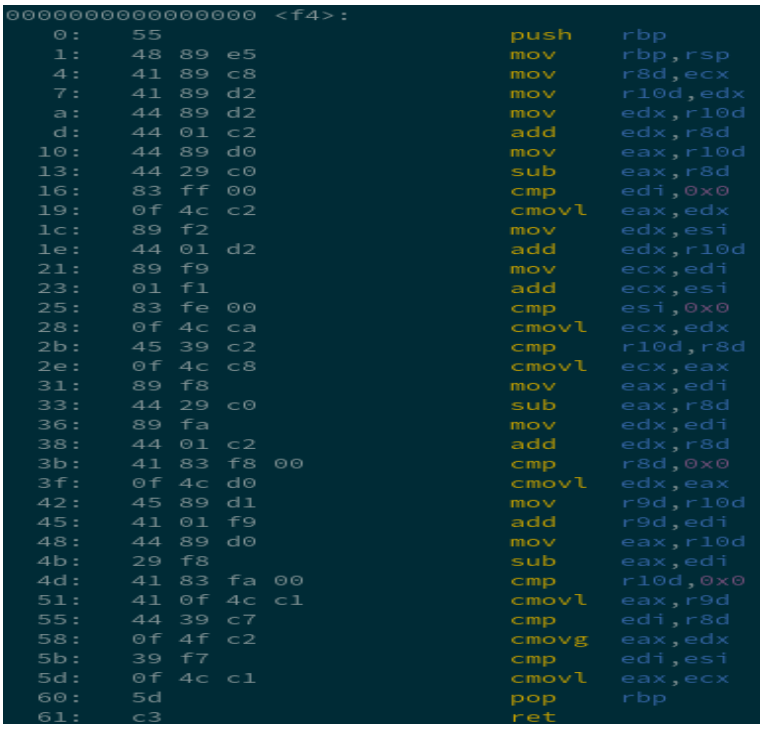
```
define dso_local i32 @f4(i32 noundef %a, i32 noundef %b, i32 noundef %c, i32 noundef %d) #0 {
entry:
  %cmp = icmp slt i32 %a, %b
  %cmp1 = icmp slt i32 %c, %d
  %cmp3 = icmp slt i32 %a, 0
  %add = add nsw i32 %c, %d
  %sub = sub nsw i32 %c, %d
  %0 = select i1 %cmp3, i32 %add, i32 %sub
  %cmp6 = icmp slt i32 %b, 0
  %add8 = add nsw i32 %b, %c
  %add10 = add nsw i32 %a, %b
  %1 = select i1 %cmp6, i32 %add8, i32 %add10
  %2 = select i1 %cmp1, i32 %0, i32 %1
  %cmp12 = icmp sgt i32 %a, %d
  %cmp14 = icmp slt i32 %d, 0
  %sub16 = sub nsw i32 %a, %d
  %add18 = add nsw i32 %a, %d
  %3 = select i1 %cmp14, i32 %sub16, i32 %add18
  %cmp20 = icmp slt i32 %c, 0
  %add22 = add nsw i32 %c, %a
  %sub24 = sub nsw i32 %c, %a
  %4 = select i1 %cmp20, i32 %add22, i32 %sub24
  %5 = select i1 %cmp12, i32 %3, i32 %4
  %6 = select i1 %cmp, i32 %2, i32 %5
  ret i32 %6
}
```

(b) LLVM IR (after transformation)

Fig. 3. test_f4 microbenchmark: LLVM IR transformation



(a) Disassembly (before transformation)



```
0000000000000000 <f4>:
   0:   55                      push   rbp
   1:   48 89 e5                mov    rbp,rsp
   4:   41 89 c8                mov    r8d,ecx
   7:   41 89 d2                mov    r10d,edx
   a:   44 89 d2                mov    edx,r10d
   d:   44 01 c2                add    edx,r8d
  10:   44 89 d0                mov    eax,r10d
  13:   44 29 c0                sub    eax,r8d
  16:   83 ff 00                cmp    edi,0x0
  19:   0f 4c c2                cmovl  eax,edx
  1c:   89 f2                   mov    edx,esi
  1e:   44 01 d2                add    edx,r10d
  21:   89 f9                   mov    ecx,edi
  23:   01 f1                   add    ecx,esi
  25:   83 fe 00                cmp    esi,0x0
  28:   0f 4c ca                cmovl  ecx,edx
  2b:   45 39 c2                cmp    r10d,r8d
  2e:   0f 4c c8                cmovl  ecx,eax
  31:   89 f8                   mov    eax,edi
  33:   44 29 c0                sub    eax,r8d
  36:   89 fa                   mov    edx,edi
  38:   44 01 c2                add    edx,r8d
  3b:   41 83 f8 00             cmp    r8d,0x0
  3f:   0f 4c d0                cmovl  edx,eax
  42:   45 89 d1                mov    r9d,r10d
  45:   41 01 f9                add    r9d,edi
  48:   44 89 d0                mov    eax,r10d
  4b:   29 f8                   sub    eax,edi
  4d:   41 83 fa 00             cmp    r10d,0x0
  51:   41 0f 4c c1             cmovl  eax,r9d
  55:   44 39 c7                cmp    edi,r8d
  58:   0f 4f c2                cmovg  eax,edx
  5b:   39 f7                   cmp    edi,esi
  5d:   0f 4c c1                cmovl  eax,ecx
  60:   5d                      pop    rbp
  61:   c3                      ret
```

(b) Disassembly (after transformation)

Fig. 4. test_f4 microbenchmark: Disassembly transformation