中国科学院软件研究所
Institute of Software Chinese Academy of Sciences

智能软件研究中心
Intelligent Software Research Center

软件所智能软件中心PLCT实验室 王鹏 实习生

2020//03/04

目 录

• 01 mcpu=c910

```
u@u-virtual-machine:~/tools/c910-llvm-master/tmp/build$ echo '1 2' | llvm-mc -disassemble -mcpu=help
Available CPUs for this target:

  amdfam10        - Select the amdfam10 processor.
  athlon          - Select the athlon processor.
  athlon-4        - Select the athlon-4 processor.
  athlon-fx       - Select the athlon-fx processor.
  athlon-mp       - Select the athlon-mp processor.
  athlon-tbird    - Select the athlon-tbird processor.
  athlon-xp       - Select the athlon-xp processor.
  athlon64        - Select the athlon64 processor.
  athlon64-sse3   - Select the athlon64-sse3 processor.
  atom            - Select the atom processor.
```

在Target/X86中实现自定义processor

mcpu=william

此processor和winchip2一样带有自定义属性

```
  skylake-avx512 - Select the skylake-avx512 processor.
  slm            - Select the slm processor.
  tremont        - Select the tremont processor.
  westmere       - Select the westmere processor.
  william        - Select the william processor.
  winchip-c6     - Select the winchip-c6 processor.
  winchip2       - Select the winchip2 processor.
  x86-64         - Select the x86-64 processor.
  yonah          - Select the yonah processor.
  znver1         - Select the znver1 processor.
  znver2         - Select the znver2 processor.

Available features for this target:

  16bit-mode                    - 16-bit mode (i8086).
  32bit-mode                    - 32-bit mode (80386).
```

mcpu=william

显示结果

```
lib/Target/X86/X86.td:def : Proc<"william",          [FeatureX87,
FeatureSlowUAMem16, Feature3DNow]>;
```

```
test/CodeGen/X86/cpus-other.ll:; RUN: llc < %s -o /dev/null -
mtriple=i686-unknown-unknown -mcpu=william 2>&1 |
FileCheck %s --check-prefix=CHECK-NO-ERROR --allow-empty
```

test/CodeGen/X86/cpus-no-x86_64.ll:; RUN: not llc < %s -o /dev/null -mtriple=x86_64-unknown-unknown -mcpu=william 2>&1 | FileCheck %s --check-prefix=CHECK-ERROR64

test/MC/X86/x86_nop.s:# RUN: llvm-mc -filetype=obj -triple=i686-pc-linux -mcpu=william %s | llvm-objdump -d - | FileCheck %s

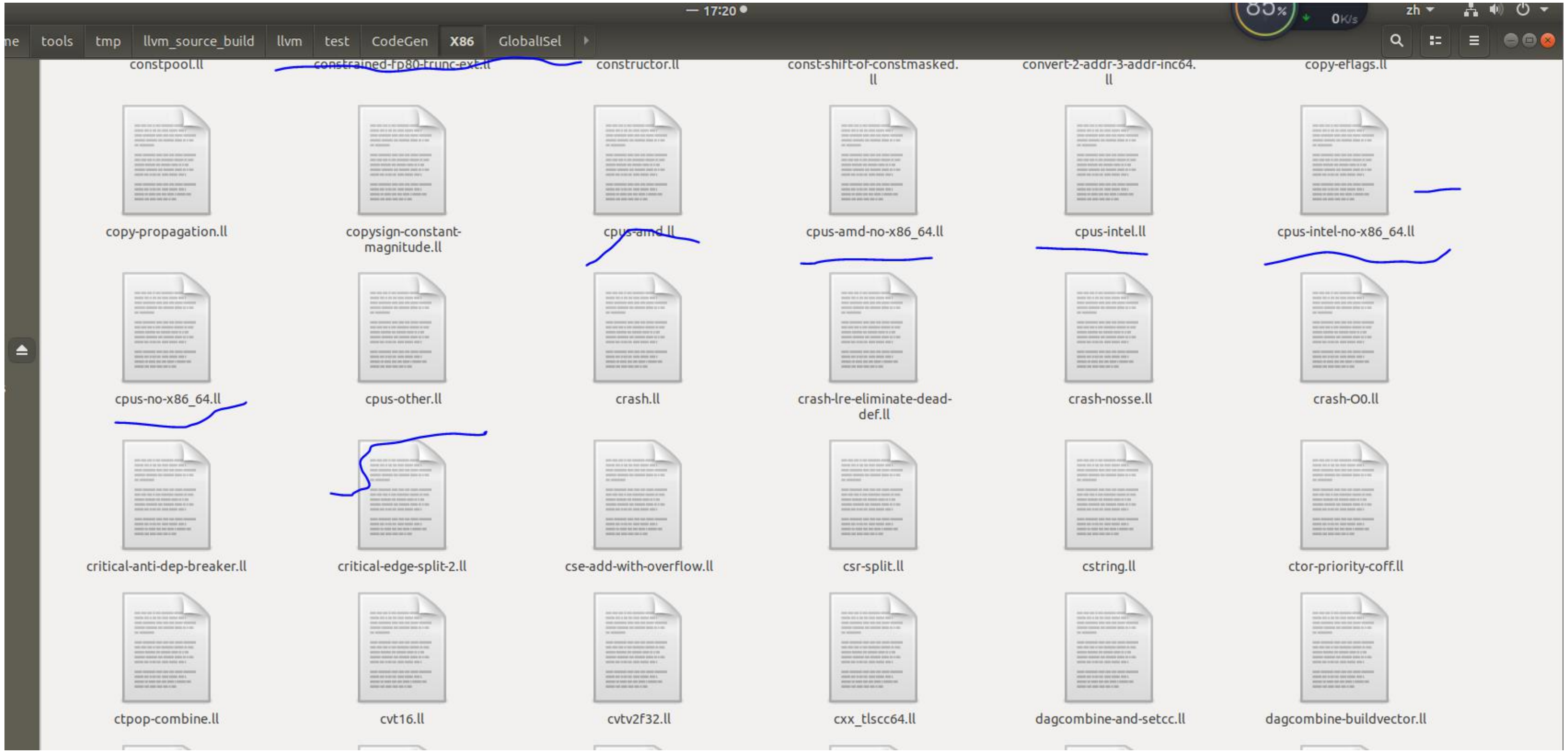
ProcessorModel的初始定义

include/llvm/Target/Target.td:

// ProcessorModel allows subtargets to specify the more general

include/llvm/Target/Target.td:

class ProcessorModel<string n, SchedMachineModel m, list<SubtargetFeature> f>


17:20
tools
tmp
llvm_source_build
llvm
test
CodeGen
X86
GlobalISel
constpool.ll
constrained-fp80-trunc-ext.ll
constructor.ll
const-shift-of-constmasked.ll
convert-2-addr-3-addr-inc64.ll
copy-eflags.ll
copy-propagation.ll
copysign-constant-magnitude.ll
cpus-amd.ll
cpus-amd-no-x86_64.ll
cpus-intel.ll
cpus-intel-no-x86_64.ll
cpus-no-x86_64.ll
cpus-other.ll
crash.ll
crash-lre-eliminate-dead-def.ll
crash-nosse.ll
crash-O0.ll
critical-anti-dep-breaker.ll
critical-edge-split-2.ll
cse-add-with-overflow.ll
csr-split.ll
cstring.ll
ctor-priority-coff.ll
ctpop-combine.ll
cvt16.ll
cvtv2f32.ll
cxx_tlscc64.ll
dagcombine-and-setcc.ll
dagcombine-buildvector.ll

很多cpu定义

u@u-virtual-machine: ~/tools/tmp/llvm_source_build/llvm/lib/Target/RISCV

File Edit View Search Terminal Help

```
//===----------------------------------------------------------------------===//

include "RISCVRegisterInfo.td"
include "RISCVCallingConv.td"
include "RISCVInstrInfo.td"
include "RISCVRegisterBanks.td"

//===----------------------------------------------------------------------===//
// RISC-V processors supported.
//===----------------------------------------------------------------------===//

def : ProcessorModel<"c910",NoSchedModel, [FeatureStdExtM, FeatureStdExtA,Feature64Bit,
                                   FeatureStdExtF, FeatureStdExtC,FeatureStdExtD]>;
def : ProcessorModel<"generic-rv32", NoSchedModel, [FeatureRVCHints]>;

def : ProcessorModel<"generic-rv64", NoSchedModel, [Feature64Bit,
                     FeatureRVCHints]>;
```

重新对mcpu=c910在RISCV目录下进行定义

```
def RISCV : Target {
  let InstructionSet = RISCVInstrInfo;
  let AssemblyParsers = [RISCVAsmParser];
  let AssemblyWriters = [RISCVAsmWriter];
  let AllowRegisterRenaming = 1;
}
//===----------------------------------------------------------------------===//
// Pfm Counters
//===----------------------------------------------------------------------===//
include "RISCVPfmCounters.td"
```

u@u-virtual-machine: ~/tools/tmp/llvm_source_build/llvm/lib/Target/RISCV

File Edit View Search Terminal Help

```
//===-- RISCVPfmCounters.td - RISCV Hardware Counters --------*- tablegen -*-===//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//===----------------------------------------------------------------------===//
//
// This describes the available hardware counters for RISCV.
//
//===----------------------------------------------------------------------===//
// RISCV Hardware Counters
def CpuCyclesPfmCounter : PfmCounter<"CYCLES">;

def DefaultPfmCounters : ProcPfmCounters {
  let CycleCounter = CpuCyclesPfmCounter;
}
def : PfmCountersDefaultBinding<DefaultPfmCounters>;
```

u@u-virtual-machine: ~/tools/tmp/llvm_source_build/llvm/test/CodeGen/RISCV

File Edit View Search Terminal Help

```
; Test that the CPU names work.
;
; First ensure the error message matches what we expect.
; CHECK-ERROR: not a recognized processor for this target

; Now ensure the error message doesn't occur for valid CPUs.
; CHECK-NO-ERROR-NOT: not a recognized processor for this target


; RUN: llc < %s -o /dev/null -mtriple=riscv64 -mcpu=c910 2>&1 | FileCheck %s --check-prefix=CHECK-NO-ERROR --allow-empty

define void @foo() {
  ret void
}
```

test/CodeGen/RISCV/cpus.ll:

```
u@u-virtual-machine: ~/tools/tmp/llvm_source_build/llvm/test/MC/RISCV
File  Edit  View  Search  Terminal  Help
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/test/MC/RISCV$ vim rv64f-valid.s
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/test/MC/RISCV$ vim rv64f-invalid.s
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/test/MC/RISCV$ vim rv64i-invalid.s
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/test/MC/RISCV$ vim rv64i-valid.s
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/test/MC/RISCV$ vim rv64d-valid.s
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/test/MC/RISCV$ vim rv64d-invalid.s
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/test/MC/RISCV$
```

test/MC/RISCV 修改支持c910 feature的汇编文件

已修改 I F D

待修改 A M C

```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ llvm-mc --version  --triple
LLVM (http://llvm.org/):
  LLVM version 10.0.0svn
  Optimized build.
  Default target: x86_64-unknown-linux-gnu
  Host CPU: skylake

  Registered Targets:
    riscv32 - 32-bit RISC-V
    riscv64 - 64-bit RISC-V
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo '1 2' | llvm-mc -disassemble -mcpu=help
llvm-mc: error: : error: unable to get target for 'x86_64-unknown-linux-gnu', see --version and --triple.
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo '1 2' | llvm-mc -disassemble -mcpu=c910
llvm-mc: error: : error: unable to get target for 'x86_64-unknown-linux-gnu', see --version and --triple.
```

```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo '1 2' | llvm-mc -triple=riscv64 --disassemble
        .text
<stdin>:1:1: warning: invalid instruction encoding
1 2
^
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo '1 2' | llvm-mc -triple=riscv64 --disassemble -mcpu=help
Available CPUs for this target:

  c910         - Select the c910 processor.
  generic-rv32 - Select the generic-rv32 processor.
  generic-rv64 - Select the generic-rv64 processor.

Available features for this target:

  64bit     - Implements RV64.
  a         - 'A' (Atomic Instructions).
  c         - 'C' (Compressed Instructions).
  d         - 'D' (Double-Precision Floating-Point).
  e         - Implements RV32E (provides 16 rather than 32 GPRs).
  f         - 'F' (Single-Precision Floating-Point).
  m         - 'M' (Integer Multiplication and Division).
  relax     - Enable Linker relaxation..
  rvc-hints - Enable RVC Hint Instructions..

Use +feature to enable a feature, or -feature to disable it.
For example, llc -mcpu=mycpu -mattr=+feature1,-feature2
        .text
<stdin>:1:1: warning: invalid instruction encoding
1 2
^
```

• 02 论文分享VTV

Enforcing Forward-Edge Control-Flow Integrity in GCC & LLVM 总结笔记

背景：攻击者目前利用堆内存损坏错误来覆盖函数指针值，然后在间接函数调用中使用，从而执行任意机器代码。这类漏洞叫前沿（front-edge）攻击，因为改变了程序控制流图（CFG）中的前沿。这篇文章介绍了前沿CFI的三种保护机制。

第一种机制是Virtual-Table验证（VTV）

它是GCC 4.9中为C ++程序实现的CFI转换，它保证了每个受保护虚拟调用使用的vtable对程序有效。vtable在只读存储器中不容易受到攻击，但是vtable调用的对象在堆上分配。攻击者会用程序中现有错误来覆盖对象中的vtable指针，并使它指向由攻击者创建的vtable。下次该对象进行虚拟调用时会使用攻击者的vtable并执行攻击者的代码。

VTV重写了进行虚拟调用的IR代码

在获得对象的vtable pointer值之后且在取消引用vtable指针之前插入验证调用。

vtable-map变量和vtable指针集

第二种机制是IFCC（Indirect Function-Call Checks）

它是在LLVM 3.4上实现的CFI转换。它可以为间接调用目标生成跳转表，然后在间接调用节点添加代码，通过转换函数指针来保护间接调用，最终保证函数指针指向跳转表。任何未指向跳转表的功能指针都被认为违反了CFI，并且被IFCC强制插入正确的表中。IFCC强制所有间接调用通过其跳转表，这样防止了无法跳转到正确类型的函数入口点的攻击。

IFCC有时会因为外部代码触发CFI违规

在IFCC插件中添加了一个标记

在IFCC转换中添加了警告模式

第三种机制是FSan（ Indirect-Call-Check Analysis）

间接调用检查分析可以帮助开发人员识别可能导致安全问题的违反CFI的行为，还可以帮助发现前沿控制流漏洞,它集成在LLVM 3.4的前端Clang的UBSan中，主要在软件开发生命周期的早期捕获CFI违规。作者将FSan用于对Chromium的应用评估，在评估过程中，FSan产生了各种未定义的行为报告。最终的解决方法是为参数提供一个void指针类型，并将强制类型转换移动到函数体中。

安全分析、性能和结论：作者分别在VTV与GCC以及IFCC与Clang编译的Chromium中进行安全分析，本次实验暂时不讨论堆栈保护。实验引入平均间接目标缩减量（Average Indirect Target Reduction，AIR）

$$AIR = \frac{1}{n}\sum_{i=1}^{n}\left(1 - \frac{T_i}{S}\right)$$

这是CFI安全性能评估标准，也就是经过CFI机制的保护后减少的可攻击目标数量，这个值越高说明防御攻击效果越好。考虑到本文是专注于保护前沿，因此用执行与AIR相同的计算front-edge AIR来度量，但平均值仅用于前边缘间接控制传输指令：间接调用和跳转。

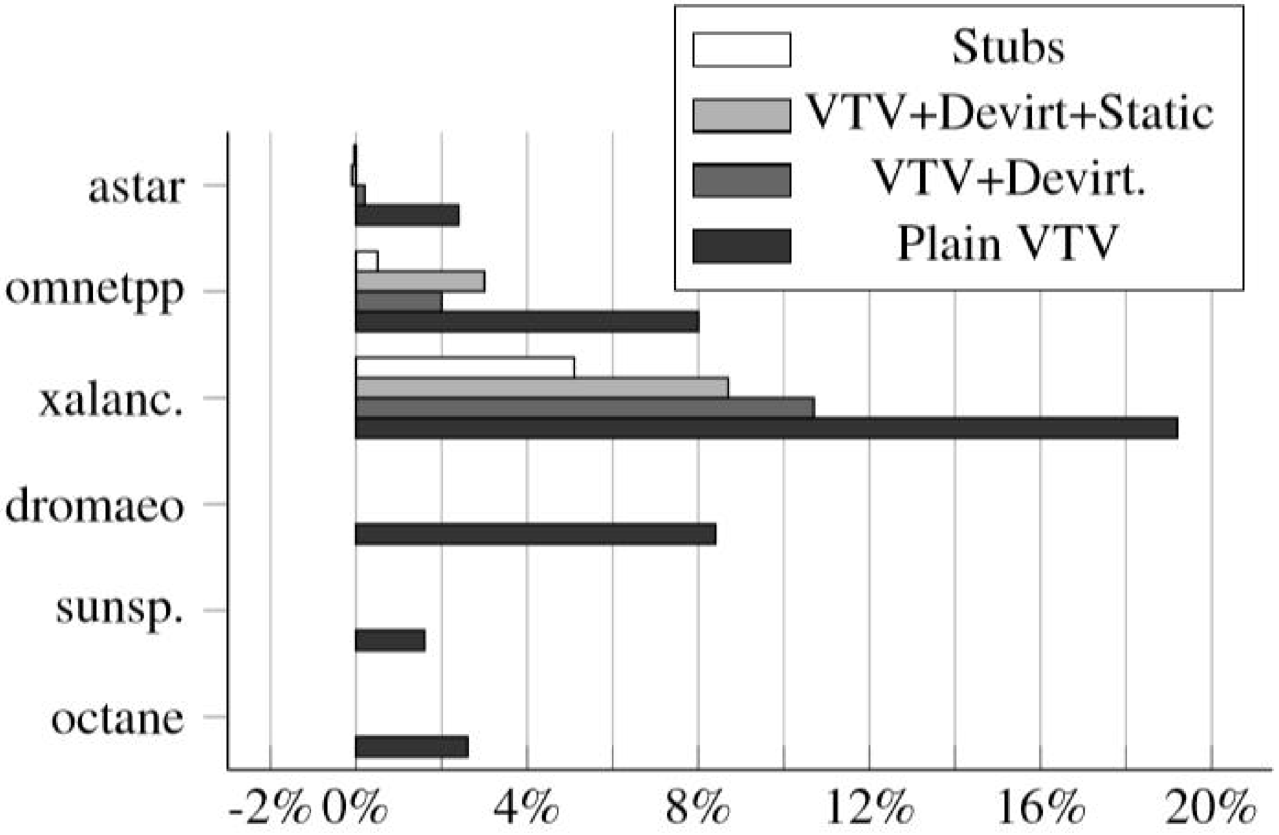


Figure 2: Relative performance overhead of VTV, with various tuning options, for the SPEC 2006 C++ benchmarks and for Chrome browser.

使用VTV与GCC编译Chromium的fAIR值是95.2%，使用IFCC与Clang编译的Chromium的fAIR值是99.8%。带有VTV的SPEC CPU 2006 C++基准测试经过配置文件引导的优化（PGO）和静态链接改进后，性能损失降低至8.7%。总体来说，这篇文章介绍的可以集成到编译器上的前沿CFI与文章中提到的其他CFI模型相比，提高了一定的性能。

```
class B {
public:
      int virtual foo ()
{...}
};

class D : public B {
public:
      int virtual foo ()
{...}
};
...
D *p1 = new D();
D *p2 = p1;   // alias
p1-> foo ();  // 1st use
...
delete (p1);  //"free"
...
p2->foo (); // BAD use!
```

内存泄漏

use after free

VTV Application


```
class B {
public:
     virtual int foo ()
{...}
};

class D : public B {
public:
     virtual int foo ()
{...}
};

B *b_ptr;
D d_obj;
b_ptr = &d_obj;

b_ptr-> foo ();
```

```
D.1 = b_ptr;
D.2 = b_ptr->_vptr.B;


D.3 = *D.2;
D.4 = call(D3 + offset)(D.1);
```

```
class B {
public:
     virtual int foo ()
{...}
};

class D : public B {
public:
     virtual int foo ()
{...}
};

B *b_ptr;
D d_obj;
b_ptr = &d_obj;

b_ptr-> foo ();
```

```
D.1 = b_ptr;
D.2 = b_ptr->_vptr.B;
D.5 = "set of valid vtable pointers
for class B";
D.6 = VerifyVtablePointer (D.5, D.2);
D.3 = *D.6;
D.4 = call(D3 + offset)(D.1);
```

vtable验证功能由标志“ -fvtable-verify =”控制

-fvtable-verify = std,

-fvtable-verify = preinit

-fvtable-verify = none

参考资料

llvm学习笔记
https://blog.csdn.net/wuhui_gdnt/article/details/62884600

[llvm-dev] [cfe-dev] When to use '-mcpu' versus '-march'
https://lists.llvm.org/pipermail/llvm-dev/2018-March/121978.html

Option Summary (GCC 选项概要)
https://blog.csdn.net/letshi/article/details/70920216
Tice2014论文

谢 谢

欢迎交流合作

2019/02/25