author Piotr Krysiuk <piotras@gmail.com> 2021-09-15 17:04:37 +0100
committer Daniel Borkmann <daniel@iogearbox.net> 2021-09-15 21:38:16 +0200
commit 37cb28ec7d3a36a5bace7063a3dba633ab110f8b (patch)
tree 14a9f73097af2955d67474fac8a8ebb0416af789
parent 356ed64991c6847a0c4f2e8fa3b1133f7a14f1fc (diff)
download linux-37cb28ec7d3a36a5bace7063a3dba633ab110f8b.tar.gz

bpf, mips: Validate conditional branch offsets

```
The conditional branch instructions on MIPS use 18-bit signed offsets
allowing for a branch range of 128 KBytes (backward and forward).
However, this limit is not observed by the cBPF JIT compiler, and so
the JIT compiler emits out-of-range branches when translating certain
cBPF programs. A specific example of such a cBPF program is included in
the "BPF_MAXINSNS: exec all MSH" test from lib/test_bpf.c that executes
anomalous machine code containing incorrect branch offsets under JIT.

Furthermore, this issue can be abused to craft undesirable machine
code, where the control flow is hijacked to execute arbitrary Kernel
code.

The following steps can be used to reproduce the issue:

  # echo 1 > /proc/sys/net/core/bpf_jit_enable
  # modprobe test_bpf test_name="BPF_MAXINSNS: exec all MSH"

This should produce multiple warnings from build_bimm() similar to:

  ------------[ cut here ]------------
  WARNING: CPU: 0 PID: 209 at arch/mips/mm/uasm-mips.c:210 build_insn+0x558/0x590
  Micro-assembler field overflow
  Modules linked in: test_bpf(+)
  CPU: 0 PID: 209 Comm: modprobe Not tainted 5.14.3 #1
  Stack : 00000000 807bb824 82b33c9c 801843c0 00000000 00000004 00000000 63c9b5ee
          82b33af4 80999898 80910000 80900000 82fd6030 00000001 82b33a98 82087180
          00000000 00000000 80873b28 00000000 000000fc 82b3394c 00000000 2e34312e
          6d6d6f43 809a180f 809a1836 6f6d203a 80900000 00000001 82b33bac 80900000
          00027f80 00000000 00000000 807bb824 00000000 804ed790 001cc317 00000001
  [...]
  Call Trace:
  [<80108f44>] show_stack+0x38/0x118
  [<807a7aac>] dump_stack_lvl+0x5c/0x7c
  [<807a4b3c>] __warn+0xcc/0x140
  [<807a4c3c>] warn_slowpath_fmt+0x8c/0xb8
  [<8011e198>] build_insn+0x558/0x590
  [<8011e358>] uasm_i_bne+0x20/0x2c
  [<80127b48>] build_body+0xa58/0x2a94
  [<80129c98>] bpf_jit_compile+0x114/0x1e4
  [<80613fc4>] bpf_prepare_filter+0x2ec/0x4e4
  [<8061423c>] bpf_prog_create+0x80/0xc4
  [<c0a006e4>] test_bpf_init+0x300/0xba8 [test_bpf]
  [<8010051c>] do_one_initcall+0x50/0x1d4
  [<801c5e54>] do_init_module+0x60/0x220
  [<801c8b20>] sys_finit_module+0xc4/0xfc
  [<801144d0>] syscall_common+0x34/0x58
  [...]
  ---[ end trace a287d9742503c645 ]---

Then the anomalous machine code executes:

=> 0xc0a18000:  addiu   sp,sp,-16
   0xc0a18004:  sw      s3,0(sp)
   0xc0a18008:  sw      s4,4(sp)
   0xc0a1800c:  sw      s5,8(sp)
   0xc0a18010:  sw      ra,12(sp)
   0xc0a18014:  move    s5,a0
   0xc0a18018:  move    s4,zero
   0xc0a1801c:  move    s3,zero

   # __BPF_STMT(BPF_LDX | BPF_B | BPF_MSH, 0)
   0xc0a18020:  lui     t6,0x8012
   0xc0a18024:  ori     t4,t6,0x9e14
   0xc0a18028:  li      a1,0
   0xc0a1802c:  jalr    t4
   0xc0a18030:  move    a0,s5
   0xc0a18034:  bnez    v0,0xc0a1ffb8           # incorrect branch offset
   0xc0a18038:  move    v0,zero
   0xc0a1803c:  andi    s4,s3,0xf
   0xc0a18040:  b       0xc0a18048
   0xc0a18044:  sll     s4,s4,0x2
   [...]

   # __BPF_STMT(BPF_LDX | BPF_B | BPF_MSH, 0)
   0xc0a1ffa0:  lui     t6,0x8012
   0xc0a1ffa4:  ori     t4,t6,0x9e14
   0xc0a1ffa8:  li      a1,0
   0xc0a1ffac:  jalr    t4
   0xc0a1ffb0:  move    a0,s5
   0xc0a1ffb4:  bnez    v0,0xc0a1ffb8           # incorrect branch offset
   0xc0a1ffb8:  move    v0,zero
   0xc0a1ffbc:  andi    s4,s3,0xf
   0xc0a1ffc0:  b       0xc0a1ffc8
   0xc0a1ffc4:  sll     s4,s4,0x2

   # __BPF_STMT(BPF_LDX | BPF_B | BPF_MSH, 0)
   0xc0a1ffc8:  lui     t6,0x8012
   0xc0a1ffcc:  ori     t4,t6,0x9e14
   0xc0a1ffd0:  li      a1,0
   0xc0a1ffd4:  jalr    t4
   0xc0a1ffd8:  move    a0,s5
   0xc0a1ffdc:  bnez    v0,0xc0a3ffb8           # correct branch offset
   0xc0a1ffe0:  move    v0,zero
   0xc0a1ffe4:  andi    s4,s3,0xf
   0xc0a1ffe8:  b       0xc0a1fff0
   0xc0a1ffec:  sll     s4,s4,0x2
   [...]

   # epilogue
   0xc0a3ffb8:  lw      s3,0(sp)
   0xc0a3ffbc:  lw      s4,4(sp)
   0xc0a3ffc0:  lw      s5,8(sp)
```

```
   0xc0a3ffc4:  lw      ra,12(sp)
   0xc0a3ffc8:  addiu   sp,sp,16
   0xc0a3ffcc:  jr      ra
   0xc0a3ffd0:  nop

To mitigate this issue, we assert the branch ranges for each emit call
that could generate an out-of-range branch.

Fixes: 36366e367ee9 ("MIPS: BPF: Restore MIPS32 cBPF JIT")
Fixes: c6610de353da ("MIPS: net: Add BPF JIT")
Signed-off-by: Piotr Krysiuk <piotras@gmail.com>
Signed-off-by: Daniel Borkmann <daniel@iogearbox.net>
Tested-by: Johan Almbladh <johan.almbladh@anyfinetworks.com>
Acked-by: Johan Almbladh <johan.almbladh@anyfinetworks.com>
Cc: Paul Burton <paulburton@kernel.org>
Cc: Thomas Bogendoerfer <tsbogend@alpha.franken.de>
Link: https://lore.kernel.org/bpf/20210915160437.4080-1-piotras@gmail.com
```

Diffstat

-rw-r--r-- arch/mips/net/bpf_jit.c 57

1 files changed, 43 insertions, 14 deletions

```
diff --git a/arch/mips/net/bpf_jit.c b/arch/mips/net/bpf_jit.c
index 0af88622c6192..cb6d22439f71b 100644
--- a/arch/mips/net/bpf_jit.c
+++ b/arch/mips/net/bpf_jit.c
@@ -662,6 +662,11 @@ static void build_epilogue(struct jit_ctx *ctx)
        ((int)K < 0 ? ((int)K >= SKF_LL_OFF ? func##_negative : func) : \
         func##_positive)

+static bool is_bad_offset(int b_off)
+{
+       return b_off > 0x1ffff || b_off < -0x20000;
+}
+
 static int build_body(struct jit_ctx *ctx)
 {
        const struct bpf_prog *prog = ctx->skf;
@@ -728,7 +733,10 @@ load_common:
                        /* Load return register on DS for failures */
                        emit_reg_move(r_ret, r_zero, ctx);
                        /* Return with error */
-                       emit_b(b_imm(prog->len, ctx), ctx);
+                       b_off = b_imm(prog->len, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_b(b_off, ctx);
                        emit_nop(ctx);
                        break;
                case BPF_LD | BPF_W | BPF_IND:
@@ -775,8 +783,10 @@ load_ind:
                        emit_jalr(MIPS_R_RA, r_s0, ctx);
                        emit_reg_move(MIPS_R_A0, r_skb, ctx); /* delay slot */
                        /* Check the error value */
-                       emit_bcond(MIPS_COND_NE, r_ret, 0,
-                                  b_imm(prog->len, ctx), ctx);
+                       b_off = b_imm(prog->len, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_bcond(MIPS_COND_NE, r_ret, 0, b_off, ctx);
                        emit_reg_move(r_ret, r_zero, ctx);
                        /* We are good */
                        /* X <- P[1:K] & 0xf */
@@ -855,8 +865,10 @@ load_ind:
                        /* A /= X */
                        ctx->flags |= SEEN_X | SEEN_A;
                        /* Check if r_X is zero */
-                       emit_bcond(MIPS_COND_EQ, r_X, r_zero,
-                                  b_imm(prog->len, ctx), ctx);
+                       b_off = b_imm(prog->len, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_bcond(MIPS_COND_EQ, r_X, r_zero, b_off, ctx);
                        emit_load_imm(r_ret, 0, ctx); /* delay slot */
                        emit_div(r_A, r_X, ctx);
                        break;
@@ -864,8 +876,10 @@ load_ind:
                        /* A %= X */
                        ctx->flags |= SEEN_X | SEEN_A;
                        /* Check if r_X is zero */
-                       emit_bcond(MIPS_COND_EQ, r_X, r_zero,
-                                  b_imm(prog->len, ctx), ctx);
+                       b_off = b_imm(prog->len, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_bcond(MIPS_COND_EQ, r_X, r_zero, b_off, ctx);
                        emit_load_imm(r_ret, 0, ctx); /* delay slot */
                        emit_mod(r_A, r_X, ctx);
                        break;
@@ -926,7 +940,10 @@ load_ind:
                        break;
                case BPF_JMP | BPF_JA:
                        /* pc += K */
-                       emit_b(b_imm(i + k + 1, ctx), ctx);
+                       b_off = b_imm(i + k + 1, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_b(b_off, ctx);
                        emit_nop(ctx);
                        break;
                case BPF_JMP | BPF_JEQ | BPF_K:
@@ -1056,12 +1073,16 @@ jmp_cmp:
                        break;
                case BPF_RET | BPF_A:
                        ctx->flags |= SEEN_A;
-                       if (i != prog->len - 1)
+                       if (i != prog->len - 1) {
                                /*
                                 * If this is not the last instruction
                                 * then jump to the epilogue
                                 */
-                               emit_b(b_imm(prog->len, ctx), ctx);
+                               b_off = b_imm(prog->len, ctx);
+                               if (is_bad_offset(b_off))
+                                       return -E2BIG;
+                               emit_b(b_off, ctx);
+                       }
                        emit_reg_move(r_ret, r_A, ctx); /* delay slot */
                        break;
                case BPF_RET | BPF_K:
@@ -1075,7 +1096,10 @@ jmp_cmp:
                        break;
                case BPF_JMP | BPF_JA:
                        /* pc += K */
-                       emit_b(b_imm(i + k + 1, ctx), ctx);
+                       b_off = b_imm(i + k + 1, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_b(b_off, ctx);
                        emit_nop(ctx);
                        break;
                case BPF_JMP | BPF_JEQ | BPF_K:
@@ -1096,7 +1120,10 @@ jmp_cmp:
                        break;
                case BPF_RET | BPF_A:
                        ctx->flags |= SEEN_A;
-                       emit_b(b_imm(prog->len, ctx), ctx);
+                       b_off = b_imm(prog->len, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_b(b_off, ctx);
                        emit_reg_move(r_ret, r_A, ctx); /* delay slot */
                        break;
                case BPF_RET | BPF_K:
@@ -1117,7 +1144,10 @@ jmp_cmp:
                        break;
                case BPF_JMP | BPF_JA:
                        /* pc += K */
-                       emit_b(b_imm(i + k + 1, ctx), ctx);
+                       b_off = b_imm(i + k + 1, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_b(b_off, ctx);
                        emit_nop(ctx);
                        break;
                case BPF_JMP | BPF_JEQ | BPF_K:
@@ -1138,7 +1168,10 @@ jmp_cmp:
                        break;
                case BPF_RET | BPF_A:
                        ctx->flags |= SEEN_A;
-                       emit_b(b_imm(prog->len, ctx), ctx);
+                       b_off = b_imm(prog->len, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_b(b_off, ctx);
                        emit_reg_move(r_ret, r_A, ctx); /* delay slot */
                        break;
                case BPF_RET | BPF_K:
```

```
                                * If this is not the last instruction
                                * then jump to the epilogue
                                */
-                               emit_b(b_imm(prog->len, ctx), ctx);
+                               b_off = b_imm(prog->len, ctx);
+                               if (is_bad_offset(b_off))
+                                       return -E2BIG;
+                               emit_b(b_off, ctx);
                                emit_nop(ctx);
                        }
                        break;
@@ -1133,8 +1157,10 @@ jmp_cmp:
                        /* Load *dev pointer */
                        emit_load_ptr(r_s0, r_skb, off, ctx);
                        /* error (0) in the delay slot */
-                       emit_bcond(MIPS_COND_EQ, r_s0, r_zero,
-                                  b_imm(prog->len, ctx), ctx);
+                       b_off = b_imm(prog->len, ctx);
+                       if (is_bad_offset(b_off))
+                               return -E2BIG;
+                       emit_bcond(MIPS_COND_EQ, r_s0, r_zero, b_off, ctx);
                        emit_reg_move(r_ret, r_zero, ctx);
                        if (code == (BPF_ANC | SKF_AD_IFINDEX)) {
                                BUILD_BUG_ON(sizeof_field(struct net_device, ifindex) != 4);
@@ -1244,7 +1270,10 @@ void bpf_jit_compile(struct bpf_prog *fp)

        /* Generate the actual JIT code */
        build_prologue(&ctx);
-       build_body(&ctx);
+       if (build_body(&ctx)) {
+               module_memfree(ctx.target);
+               goto out;
+       }
        build_epilogue(&ctx);

        /* Update the icache */
```