SAMSUNG

Linux Plumbers Conference 2018

eBPF-based tracing tools under 32 bit architectures

Maciej Słodczyk <m.slodczyk2@partner.samsung.com>
Adrian Szyndela <adrian.s@samsung.com>
Samsung R&D Institute Poland

Nov 15, 2018

The scope

Tizen OS

- SoC dependent, 32 or 64 bit arm kernel
- 32 bit userspace

Dbus Observability Tools

- low overhead, attached on the fly
- deep, accurate tracing in production environments
- BCC on top of [ku]probes powered eBPF seems perfect for that, but...

Problem

- BCC targets 64 bit systems (produces different bytecode on 32 and 64 bit host architecture)
- Unable to run on 32 bit arm userspace

Problems

LLVM compilation stages

1. „C” -> restricted C
2. restricted C -> IR (local architecture: arm)
3. IR -> eBPF (target architecture: eBPF)

Different target architectures

- step 2 – IR is 32 bit (userspace architecture)
- step 3 – IR is treated as 64 bit (eBPF default)
- architecture-dependent types size mismatch

Issues

- Verifier rejects – memset()/sizeof() based operations
- Verifier accepts a false positive - calculating offsets in userspace structures

Proposed approach

BCC

- set target architecture to eBPF at stage 2
- define sets of arch-dependent basic types and use them in eBPF program
  - can’t use native types for peeking into userspace data
  - alter native types (uint, int, ptr, long, ulong) with fixed length types (uint_t, int_t, ptr_t, long_t, ulong_t):

    ```
    typedef uint[32|64]_t uint_t
    ```
  - cons: developer won’t be able to use original platform headers
- instruct LLVM to use correct set of types based on host architecture at the moment of compilation

TODO list

- **discuss how the community wants it to be done**
- BCC: ensure upstream BCC works on 32 bit platforms
- arm64 kernel: attaching uprobes in 32 bit userspace – patchset posted, ongoing review
- arm kernel: lack of support for Thumb instructions probing

Thank You

SAMSUNG

Appendix

Test program (C):

```
#define offsetof(TYPE, MEMBER)        ((unsigned
int)&((TYPE *)0)->MEMBER)

int bpf_probe_read(void *dst, int size, void *src);

struct pt_regs {
        unsigned long uregs[18];
};

struct teststruct
{
        const char *field;
};

int test_ok(struct pt_regs *ctx, void *conn, struct
teststruct *t) {
        const char *c = t->field;
        return *c;
}
```

Test program (restricted C):

```
#define offsetof(TYPE, MEMBER)        ((unsigned int)&((TYPE *)0)->MEMBER)

int bpf_probe_read(void *dst, int size, void *src);

struct pt_regs {
        unsigned long uregs[18];
};

struct teststruct
{
        const char *field;
};

int test_ok(struct pt_regs *ctx) {
        struct teststruct *t = ctx->uregs[0];
    const char *c = ({
          typeof(void*) _val;
          __builtin_memset(&_val, 0, sizeof(_val));
          bpf_probe_read(&_val, sizeof(_val), (char *)t
                         + offsetof(struct teststruct, field));
          _val; });
     return ({
          typeof(const char) _val;
          __builtin_memset(&_val, 0, sizeof(_val));
          bpf_probe_read(&_val, sizeof(_val), (char *)c);
          _val; });
}
```

Appendix

IR – target is 32 bit ARM

```
; ModuleID = 'tc-rewritten.c'
source_filename = "tc-rewritten.c"
target datalayout = "e-m:e-p:32:32-i64:64-v128:64:128-a:0:32-n32-S64"
target triple = "armv7-unknown-linux-gnueabi"

%struct.pt_regs = type { [18 x i32] }
%struct.teststruct = type { i8* }

; Function Attrs: noinline nounwind optnone
define i32 @test_ok(%struct.pt_regs* %ctx) #0 {
entry:
  %ctx.addr = alloca %struct.pt_regs*, align 4
  %t = alloca %struct.teststruct*, align 4
  %c = alloca i8*, align 4
  %_val = alloca i8*, align 4
  %tmp = alloca i8*, align 4
...
  %3 = bitcast i8** %_val to i8*
  call void @llvm.memset.p0i8.i32(i8* %3, i8 0, i32 4, i32 4, i1 false)
  %4 = bitcast i8** %_val to i8*
  %5 = load %struct.teststruct*, %struct.teststruct** %t, align 4
  %6 = bitcast %struct.teststruct* %5 to i8*
  %add.ptr = getelementptr inbounds i8, i8* %6, i32 0
  %call = call i32 @bpf_probe_read(i8* %4, i32 4, i8* %add.ptr)
  %7 = load i8*, i8** %_val, align 4
  store i8* %7, i8** %tmp, align 4
  %8 = load i8*, i8** %tmp, align 4
  store i8* %8, i8** %c, align 4
  call void @llvm.memset.p0i8.i32(i8* %_val1, i8 0, i32 1, i32 1, i1 false)
  %9 = load i8*, i8** %c, align 4
  %call2 = call i32 @bpf_probe_read(i8* %_val1, i32 1, i8* %9)
...
  ret i32 %conv
}
```

IR – target is BPF

```
; ModuleID = 'tc-rewritten.c'
source_filename = "tc-rewritten.c"
target datalayout = "e-m:e-p:64:64-i64:64-n32:64-S128"
target triple = "bpf"

%struct.pt_regs = type { [18 x i64] }
%struct.teststruct = type { i8* }

; Function Attrs: noinline nounwind optnone
define i32 @test_ok(%struct.pt_regs* %ctx) #0 {
entry:
  %ctx.addr = alloca %struct.pt_regs*, align 8
  %t = alloca %struct.teststruct*, align 8
  %c = alloca i8*, align 8
  %_val = alloca i8*, align 8
  %tmp = alloca i8*, align 8
...
  %3 = bitcast i8** %_val to i8*
  call void @llvm.memset.p0i8.i64(i8* %3, i8 0, i64 8, i32 8, i1 false)
  %4 = bitcast i8** %_val to i8*
  %5 = load %struct.teststruct*, %struct.teststruct** %t, align 8
  %6 = bitcast %struct.teststruct* %5 to i8*
  %add.ptr = getelementptr inbounds i8, i8* %6, i64 0
  %call = call i32 @bpf_probe_read(i8* %4, i32 8, i8* %add.ptr)
  %7 = load i8*, i8** %_val, align 8
  store i8* %7, i8** %tmp, align 8
  %8 = load i8*, i8** %tmp, align 8
  store i8* %8, i8** %c, align 8
  call void @llvm.memset.p0i8.i64(i8* %_val1, i8 0, i64 1, i32 1, i1
false)
  %9 = load i8*, i8** %c, align 8
  %call2 = call i32 @bpf_probe_read(i8* %_val1, i32 1, i8* %9)
...
  ret i32 %conv
}
```