New developments in the SFrame stack trace format

Indu Bhagat (Oracle)
Jose E. Marchesi (Oracle)

Tracing Summit 2023

Agenda

- Brief History of SFrame
- Motivation behind SFrame
  - Fast, low-overhead stack tracing
- Introduction to the SFrame format
- New developments since SFrame V1
- Ongoing and future work

Brief History of SFrame

- The **S**imple **Frame** stack trace format
- [January’23] SFrame V1 released with GNU binutils 2.40
- [May’23] POC of SFrame-based user space stack unwinder in the Linux kernel
- [July’23] SFrame V2 released with GNU binutils 2.41

Stack traces

- Stack traces are needed for all profiling, tracing and debugging tools, and more
- What methods are used to generate stack traces?
  - [Heuristics] Decode and Infer stack ops
  - [Dedicated Reg / HW] Frame pointer method, LBR
  - [Debug Format] EH Frame, Application-specific formats (ORC etc.)

Stack traces – Current methods

Method	Pros	Cons
Frame pointer	Simple, fast	Performance impact; Applications may not have preserved frame pointer
EH Frame	Versatile	Complex unwinder with high resource requirements
ORC, and other application-specific formats	Fast, “off-band”	Not supported in toolchain. Need reverse engineering of binaries

Key requirements of an effective stack trace format

- Requirements for fast, low-overhead stack tracing:
  - Support for asynchronous stack tracing
  - Low overhead stack tracing
  - Low complexity stack tracer
  - Generated by the Toolchain
- SFrame format has been designed to fulfill these requirements

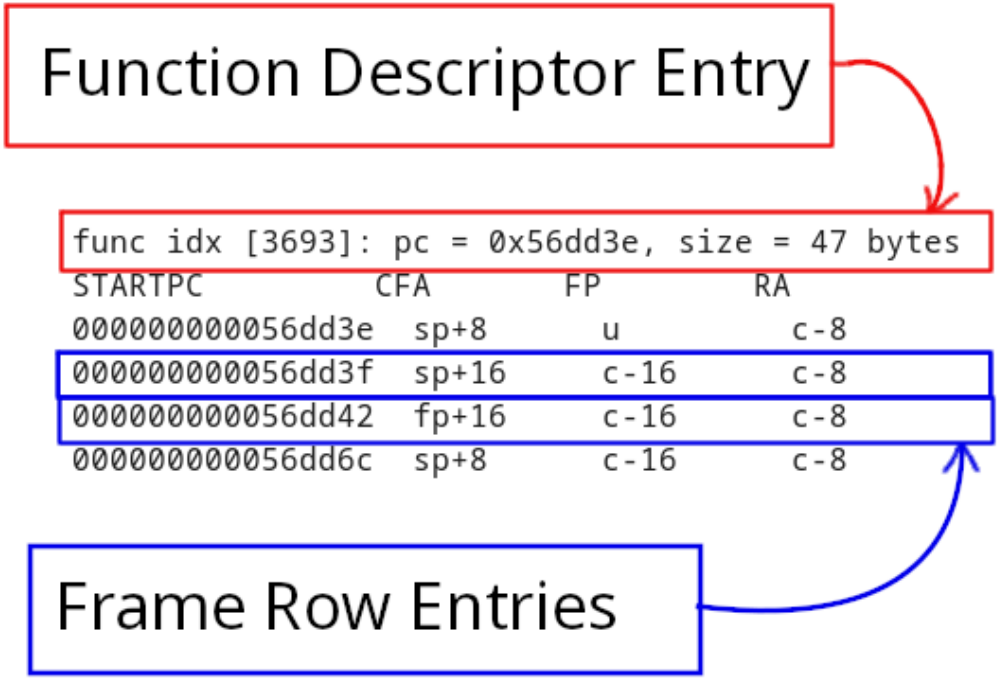
SFrame – Simple Frame stack trace format

- First defined and implemented in Binutils 2.40
  - [Spec] https://sourceware.org/binutils/docs/
- Encodes the minimal necessary information required to stack trace, per PC:
  - Canonical Frame Address (CFA)
  - Frame Pointer (FP)
  - Return Address (RA)

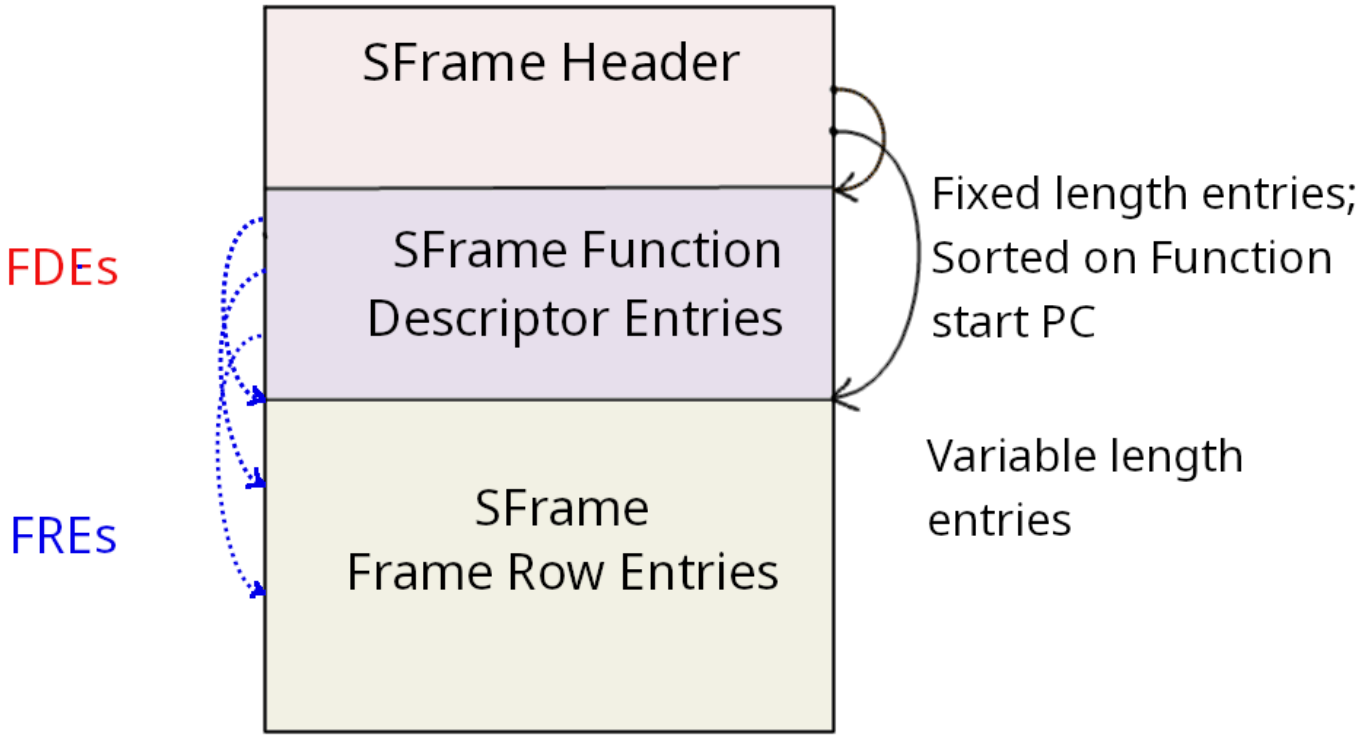
SFrame – overview

- Current version: SFRAME_VERSION_2
- New ELF section named ‘.sframe’ in a segment of its own, PT_GNU_SFRAME
  - Use --gsframe to GNU assembler (as)
- Defined for x86_64 (AMD64) and aarch64 (AAPCS64) ABIs
  - Adding more ABIs will need format revision
- Has support for pltN entries, PAC-related RA signing constructs

SFrame – Stack trace info per function



SFrame – overall data layout



SFrame – FDE representation

- SFrame Function Descriptor Entry (FDE)
  - Function start PC
  - Function size in bytes
  - Type of code block (regular or pltN)
  - Offset to the SFrame FREs
  - Number and **Type** of FREs (a.k.a. FRE encoding)

SFrame – FRE representation

- SFrame Frame Row Entry (FRE)
  - Backbone of SFrame stack trace information
  - “Given a PC, what are the stack offsets to recover the CFA, FP and RA"
- FRE contains
  - Start IP offset (a.k.a, offset from the start PC of function) encoded in 1 / 2 / 4 bytes
  - Variable number of stack offsets
  - Size of stack offsets is tunable

SFrame – What makes it effective

- Generated by the Toolchain
- Simple format designed with fast, low-overhead stack tracing in mind
  - Let’s talk about its three key features...

SFrame – Three key features - (1/3)

- FDEs are sorted on start PCs of functions
  - Quickly find the stack trace data for the PC
  - Stack tracers can use binary search to find the FDE
  - FDE holds the offset to where the corresponding SFrame FREs

SFrame – Three key features - (2/3)

- Stack offsets to recover CFA, RA, FP are encoded directly in the FRE
  - No complex expressions, no stack machine needed to generate stack offsets

SFrame – Three key features - (3/3)

- On-disk FRE representation has some space-saving strategies
  - Compactness is important
- Space-efficient on-disk encoding is necessary
  - Functions are of varied sizes
  - Each function uses stack differently

SFrame stack trace generation is easy

```
/* Find the SFrame FRE, given the PC.  */
sframe_fre fre;
pc -= sframe_vma;
err = sframe_find_fre(sfsec, pc, &fre);
/* Get the CFA offset from the FRE.  */
cfa_offset = sframe_fre_get_cfa_offset(sfsec, fre, &err);
cfa = ((sframe_fre_get_base_reg_id(fre, &err) == SFRAME_BASE_REG_SP)
       ? sp : fp) + cfa_offset;
/* Get the RA offset from the FRE.  */
ra_offset = sframe_fre_get_ra_offset(sfsec, fre, &err);
ra_stack_loc = cfa + ra_offset;
return_addr = *ra_stack_loc;
/* Get the FP offset from the FRE.  */
rfp_offset = sframe_fre_get_fp_offset(sfsec, fre, &err);
rfp_stack_loc = cfa + rfp_offset;
fp = *rfp_stack_loc;

/* Prepare for next iteration.  */
rsp = cfa;
pc = return_addr;
```

SFrame format – What's next?

- [GNU as] Directive .cfi_escape are not handled
  - Not fully asynchronous, but close
- [Not supported] Using DRAP to realign stack

```
leaq    8(%rsp), %r10
andq    $-16, %rsp
pushq   -8(%r10)
pushq   %rbp
movq    %rsp, %rbp
```

- Support use-cases of the SFrame format
  - Linux kernel, User space applications, ...

Changes in V2

- Enhancement: Size of pltN Entry is encoded explicitly
- Bugfix: SFrame FDE being 17 bytes, caused misaligned accesses in libsframe
  - SFrame FDE size is now 20 bytes; including 2 trailing empty bytes
- Other toolchains should ideally prefer V2

User space stack tracing in Linux kernel

- Relieve user space applications from the need to be built with frame-pointer preserved
- Fast, low-overhead stack tracing
  - Simple unwinder

User space stack tracing in Linux kernel

- [POC] SFrame based stack tracer for user space on linux-toolchains@vger.kernel.org
  - New Kconfig option USER_UNWINDER_SFRAME
  - Add to task_struct: struct sframe_state *sframe_state;
    - sframe_state_setup () in load_elf_binary ()
  - small library of SFrame decode and access APIs, stack tracer
    - Other helper routines like iterate_phdr ()
  - Changes made directly in perf_callchain_user()
    - perf, bpf_get_stack (), DTrace

Issues with the POC

- Accessed SFrame data in NMI context
- sframe_callchain_user() hooked into perf_callchain_user()
- Discussed next steps at LSF/MM/BPF Summit (May 2023)
  - SFrame, Steve Rostedt, Indu Bhagat

Brief discussion notes - I

- Changes in perf
  - “Work to do before return-to-user”: Get the stack trace on the return-to-user path (ptrace () path) in Kernel context
  - Set state to indicate that “user space stack trace will be added later”
- User space unwinder
  - Rework the interfaces
  - “Something that perf calls into, not hooked into perf”

Brief discussion notes - II

- We need to be able to track dlopen/dlclose, or additional shared libraries loaded via the dynamic linker at the task execution time.
- Notes https://lwn.net/Articles/932209/

Summary

- The impact of SFrame format
- Recent new developments
  - SFrame V2
  - User space stack tracing in Linux kernel
- Get in touch
  - linux-toolchains@vger.kernel.org
  - binutils@sourceware.org

New developments in the SFrame stack trace format

~ Q & A ~