Contribution ID: **239** Type: **not specified**

Enabling user mode programs to emit into trace_event / dyn_event

Wednesday, September 22, 2021 7:30 AM (25 minutes)

Summary
We have many user processes today that run in various locations and control groups. To know every binary location for each version becomes a challenge. We also have common events that trace out among many processes. This makes using uprobes a challenge, but not impossible. However, having a way for user processes to publish data directly to trace_events enables a much easier path toward collecting and analyzing all of this data. We do not need to track per-binary locations, nor do we need to enter the control groups to find the real binary paths.

Today the main way to create and get data into a trace_event from a user mode program is by using uprobes. Uprobes require the locations of each binary that wants to be traced in addition to all of the argument locations. We propose an alternative mechanism which allows for faster operation and doesn't require knowing code locations. While we could use inject and dynamic_events to do this as well, user processes don't have a way to know when inject should be written to.

Knowing when to trace
In order to have good performance, user mode programs must know when an event should be traced. Uprobes do this via a nop vs int3 and handle the break point in the die chain handler. To account for this a tracefs file called user_events_mmap will be created which will be mmap'd in each user process that wants to emit events. Each byte in the mmap data will represent 0 if nothing is attached to the trace_event, and non-zero if there is. It would be nice to use each bit of the byte to represent what system is attached (IE: Bit 0 for ftrace, bit 1 for perf, etc). This has the limitation however of only being able to support up to 8 systems, unless bit 7 is reserved for "other". User programs simply branch on non-zero to determine if anything wants tracing. To protect the system from running out of trace_events the amount of user defined events is limited to a single page. The kernel side keeps the page updated via the underlying trace_events register callbacks. The page is shared across all processes, it's mapped in as read only upon the mmap syscall.

Opening / Registering Events
Before a program can write events they need to register/open events. To do this an IOCTL is issued to a tracefs file called user_events_data with a payload describing the event. The return value of the IOCTL represents the byte within the mmap data to use to check if tracing is enabled or not. The open file can now be used to write data for the event that was just registered. A matching IOCTL is available to delete events, delete is only valid when all references have been closed.

Writing Event Data
Writing event data using the above file is done via the write syscall. The data described in each write call will represent the data within the trace_event. The kernel side will map this data into each system that is registered, such as ftrace, perf and eBPF automatically for the user.

Event status pseudo code:

```
page\_fd = open("/sys/kernel/tracing/user\_events\_mmap");
status\_page = mmap(page\_fd, PAGE\_SIZE);
close(page\_fd);
```

Register event pseudo code:

```
event\_fd = open("/sys/kernel/tracing/user\_events\_data");
event\_id = IOCTL(event\_fd, REG, ``MyUserEvent");
```

Write event pseudo code:

```
if (status\_page[event\_id]) write(event\_fd, ``My user payload");
```

Delete event pseudo code:

```
IOCTL(event\_fd, DEL, ``MyUserEvent");
```

I agree to abide by the anti-harassment policy

I agree

Primary author: BELGRAVE, Beau

Presenter: BELGRAVE, Beau

Session Classification: Tracing MC

Track Classification: Tracing MC