LSM BPF Programs

These BPF programs allow runtime instrumentation of the LSM hooks by privileged users to implement system-wide MAC (Mandatory Access Control) and Audit policies using eBPF.

Structure

The example shows an eBPF program that can be attached to the `file_mprotect` LSM hook:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\(linux-master)(Documentation)(bpf)prog_lsm.rst, line 18)**
>
> Unknown directive type "c:function".
>
> ```
> .. c:function:: int file_mprotect(struct vm_area_struct *vma, unsigned long reqprot, unsigned long pr
> ```

Other LSM hooks which can be instrumented can be found in `include/linux/lsm_hooks.h`.

eBPF programs that use Documentation/bpf/btf.rst do not need to include kernel headers for accessing information from the attached eBPF program's context. They can simply declare the structures in the eBPF program and only specify the fields that need to be accessed.

```
struct mm_struct {
        unsigned long start_brk, brk, start_stack;
} __attribute__((preserve_access_index));

struct vm_area_struct {
        unsigned long start_brk, brk, start_stack;
        unsigned long vm_start, vm_end;
        struct mm_struct *vm_mm;
} __attribute__((preserve_access_index));
```

> **Note**
>
> The order of the fields is irrelevant.

This can be further simplified (if one has access to the BTF information at build time) by generating the `vmlinux.h` with:

```
# bpftool btf dump file <path-to-btf-vmlinux> format c > vmlinux.h
```

> **Note**
>
> `path-to-btf-vmlinux` can be `/sys/kernel/btf/vmlinux` if the build environment matches the environment the BPF programs are deployed in.

The `vmlinux.h` can then simply be included in the BPF programs without requiring the definition of the types.

The eBPF programs can be declared using the``BPF_PROG`` macros defined in tools/lib/bpf/bpf_tracing.h. In this example:

- `"lsm/file_mprotect"` indicates the LSM hook that the program must be attached to
- `mprotect_audit` is the name of the eBPF program

```
SEC("lsm/file_mprotect")
int BPF_PROG(mprotect_audit, struct vm_area_struct *vma,
             unsigned long reqprot, unsigned long prot, int ret)
{
        /* ret is the return value from the previous BPF program
         * or 0 if it's the first hook.
         */
        if (ret != 0)
                return ret;

        int is_heap;

        is_heap = (vma->vm_start >= vma->vm_mm->start_brk &&
                   vma->vm_end <= vma->vm_mm->brk);

        /* Return an -EPERM or write information to the perf events buffer
         * for auditing
         */
        if (is_heap)
                return -EPERM;
}
```

The `__attribute__((preserve_access_index))` is a clang feature that allows the BPF verifier to update the offsets for the access at runtime using the Documentation/bpf/btf.rst information. Since the BPF verifier is aware of the types, it also validates all the

accesses made to the various types in the eBPF program.

Loading

eBPF programs can be loaded with the :manpage:`bpf(2)` syscall's `BPF_PROG_LOAD` operation:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\(linux-master)(Documentation)(bpf)prog_lsm.rst, line 98); *backlink***
>
> Unknown interpreted text role "manpage".

```
struct bpf_object *obj;

obj = bpf_object__open("./my_prog.o");
bpf_object__load(obj);
```

This can be simplified by using a skeleton header generated by `bpftool`:

```
# bpftool gen skeleton my_prog.o > my_prog.skel.h
```

and the program can be loaded by including `my_prog.skel.h` and using the generated helper, `my_prog__open_and_load`.

Attachment to LSM Hooks

The LSM allows attachment of eBPF programs as LSM hooks using :manpage:`bpf(2)` syscall's `BPF_RAW_TRACEPOINT_OPEN` operation or more simply by using the libbpf helper `bpf_program__attach_lsm`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\(linux-master)(Documentation)(bpf)prog_lsm.rst, line 120); *backlink***
>
> Unknown interpreted text role "manpage".

The program can be detached from the LSM hook by *destroying* the `link` link returned by `bpf_program__attach_lsm` using `bpf_link__destroy`.

One can also use the helpers generated in `my_prog.skel.h` i.e. `my_prog__attach` for attachment and `my_prog__destroy` for cleaning up.

Examples

An example eBPF program can be found in tools/testing/selftests/bpf/progs/lsm.c and the corresponding userspace code in tools/testing/selftests/bpf/prog_tests/test_lsm.c