Modernizing Android BPF
& The Android BPF Security Model
Neill Kapron <nkapron@google.com>
The State of BPF in Android

- Outdated libraries—libbpf, bcc, bpftool
- Custom/curated library for BPF program development
- Limited functionality enabled
  - No CO-RE
  - Few helpers
  - No bpftrace
Android BPF Goals

- Enable Modern BPF functionality
  - CO-RE
  - Libbpf helpers
- Enable secure vendor access to BPF tracepoints
- Build solid foundation for future use cases
Android Concepts

- Bionic standard C library
- Each user+application combination has a dedicated UID
- Two Kernel branches for each release
  - Android14-5.15, Android14-6.1
- Older devices can upgrade to new OS with older kernels
- ACK - Android Common Kernel
- GKI - Generic Kernel Image
- KMI - Kernel ABI Stability maintained within kernel branches
  - Android14-5.15, Android14-6.1, etc
- Trusted file systems - dmverity
- Supports armv7, aarch64, x86, x86_64, riscv64
The State of BPF in Android

Android Concepts
Android BPF Security
Current Implementation
CO-RE + Access Control
Questions

Debugging & Development
- Development, Performance, & Test Engineers

Release Telemetry & Functionality
- Networking/Tethering/Bandwidth Measurement
- System
- SOC & Device Manufacturers - ‘Vendors’
Current Android BPF Security Restrictions

- Limit loading capabilities to a single system program
  - (now separate loader for networking)
  - Partially due to previous requirement for CAP_SYS_ADMIN
- Loader program is one-shot (exits upon loading programs in early init)
- Selinux restrictions
- Control BPF program attach points.
- BPF Program types restricted based on source
- Certain hooks (fentry/fexit) must remain disabled
The State of BPF in Android
Android Concepts
Android BPF Security
Current Implementation
CO-RE + Access Control
Questions
Android’s File Systems Relevant to BPF

The State of BPF in Android
Android Concepts
Android BPF Security
Current Implementation
CO-RE + Access Control
Questions
## Supported Program Types

<table>
<thead>
<tr>
<th>BPF Program Types</th>
<th>Networking</th>
<th>System</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGROUP_SKB</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGROUP_SOCK</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGROUP_SOCK_ADDR</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KPROBE</td>
<td></td>
<td>Y</td>
<td>R</td>
</tr>
<tr>
<td>PERF_EVENT</td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>SCHED_ACT</td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>SCHED_CLS</td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>SOCKET_FILTER</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>TRACEPOINT</td>
<td></td>
<td>Y</td>
<td>R</td>
</tr>
<tr>
<td>XDP</td>
<td></td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

Y = Supported, R = Requested
Challenges

- Boot time
- Memory overhead
- Kernel/Userspace/BPF object ABI compatibility
- Security
Possibilities for Enabling CO-RE in Android

- Implement custom Android-Specific library
  - Constant development and maintenance required as new BPF features are created
  - Can optimize for our specific use case
- Integrate Libbpf into existing bpfloader
  - Does not solve boot time problem
  - Potential for significant increase in memory usage
  - Potential problems with compatibility between vendor BPF programs and system libbpf library version
- Enable BPF programs to use libbpf natively
  - Allows developers and vendors to choose when their programs are loaded
  - Resolves compatibility issue between system libraries and vendor bpf programs
  - Requires additional work to develop access control mechanism
- Other approaches?
Attach Point Access Control

- Need to verify that tracepoints are part of KMI before attaching
  - KMI varies between kernel branches
  - KMI additions can occur post kernel release (requires allowlist updatability)
- Could be accomplished via allowlist in bpfloader
  - Check bpf program’s attach points prior to loading into kernel
  - Allowlist must be dynamic and maintain support for all kernel versions
- BPF Program/Kernel Module based access control
  - Add hooks into bpf_prog_load() and bpf_prog_attach() functions
  - First BPF program loaded as part of boot
  - Check subsequent bpf progs against running kernel’s KMI
  - Enables the control of ‘native’ libbpf programs
  - Unknown- how to check source of bpf program in kernel?
Attach Point Access Control

BPF Loader Allowlist Approach

The State of BPF in Android
Android Concepts
Android BPF Security
Current Implementation
CO-RE + Access Control
Questions
Attach Point Access Control

“Native libbpf” + BPF access control program

The State of BPF in Android
Android Concepts
Android BPF Security
Current Implementation
CO-RE + Access Control
Questions
BPFloader Open Questions

- What is the compatibility story for libbpf?
  - Do we need a trampoline library for future API changes?
- What can be done to optimize loading at boot time?
- Can system BTF data be cached by loader process?
  - Refactor libbpf calls to allow passing in BTF object
- Do we need to extend metadata for selinux policy?
‘Native Libbpf’ Open Questions

- Will this approach pass security review?
- How do we get KMI ACL from kernel
  - Do we create a subset of KMI?
- How to pair BPF object with filesystem source for verification?
- What can be done to optimize BTF memory footprint?
Thank You!
Neill Kapron <nkapron@google.com>