A tale of two eBPFs: Towards Managing eBPF Programs for the Rest of US

Theophilus A. Benson
Carnegie Mellon University
About Me...

Professor @ ECE--CMU
• Addressing digital divide in global south
• Management frameworks for the cloud
• Web performance
Facebook, Google, Isovalent, Microsoft, and Netflix announce eBPF Foundation

Cloudflare architecture and how BPF eats the world

05/18/2019

Marek Majewski

Recently at Netflix, I gave a talk titled "Linux and Cloudflare: a love story that matters the quiz..." Here is a transcript of the talk:

The Silent Platform Revolution: How eBPF Is Fundamentally Transforming Cloud-Native Platforms
Security/Monitoring (Emerging)
Falco, Tetragon, Pixie, **DDoS:** Cloudflare, FW: Cloudflare-Rakelimit,

Observability/Monitoring
Falco, Hubble, Sysdig, Tracee, RH OCP(Netobserv), Flowmill (not open-source), SkyDive, DataDog, Elastic

Networking:
FB-Katran, Cloudflare, Cilium, Isovalent-Cilium, Calico, Cloudflare-Rakelimit, Huawei-Mizar, K8s-envoy,
Popular Use cases Bias Our Views

- Stable Code Bases
- Small number of same Programs Deployed Everywhere
- Generally Understood/trusted Code
- No need to Write/Extend code

- Testing Frameworks
- Orchestration
- Understand code
- Extend/Edit Code

- Debugging
- Policy Enforcement
Ecosystem of DevTools to enable safe and quick incorporation/extension of third party eBPF code at scale
Popular Use cases Bias Our Views

- Stable Code Bases
- Small number of same Programs Deployed Everywhere
- Generally Understood/trusted Code
- No need to Write/Extend code

- Testing Frameworks
- Orchestration
- Understand code
- Extend/Edit Code

- Debugging
- Policy Enforcement
Roadmap

- **Background**
- **Understanding Code**
  
  (Registry)
- **Extending Code**
  
  (OPENED)
- **Orchestrating @Scale**
  
  (Apiary)
Application Deployments – 10 Years ago.
BPF: A New Type of Software

02 Dec 2019

At Netflix we have 15 BPF programs running on cloud servers by default; Facebook has 40. These programs are not processes or kernel modules, and don't appear in traditional observability tools. They are a new type of software, and make a fundamental change to a 50-year old kernel model by introducing a new interface for applications to make kernel requests, alongside syscalls.
Key eBPF Primitives and Abstractions

**eBPF Program:** Code pair + data structures

**Hookpoints:** where to attach eBPF programs

Kernel: eBPF Program
User space: Control
Key eBPF Primitives and Abstractions

BPF: A New Type of Software

02 Dec 2019

At Netflix we have 15 BPF programs running on cloud servers by default; Facebook has 40. These programs are not processes or kernel modules, and don’t appear in traditional observability tools. They are a new type of software, and make a fundamental change to a 50-year old kernel model by introducing a new interface for applications to make kernel requests, alongside syscalls.
Introduce Mismatch between Application and Reality
Introduce Mismatch between Application and Reality

Introduces unexpected and unpredictable performance variability
Introduce Mismatch between Application and Reality

Introduces unexpected and unpredictable performance variability

Load Balancer

Explain/Understand Third party Code

Manage deployment on thousands of Servers
Roadmap

- **Background**
- **Understanding Code (Registry)**
- **Extending Code (OPENED)**
- **Orchestrating @Scale (Apiary)**
Static Analysis

- Precise and Correct
- Hard to understand
- Fails to scale

Machine Learning (LLMs)

- Potentially introduces errors
- Easy to understand
- Arbitrary scaling properties
Input Granularity

Hyper Params

Too Verbose

Too abstract
Hyper Params

Per line summary
(too mundane)

Per file summary
(too convoluted)

Input Granularity
Hyper Params = func (prog length)

Explainability is on a Per Function Basis

Decompose into functions
(eBPF)Registry: Simplify Code Introspection and Analysis

- Automated inference of code documentation

- Automated detection of code requirements
  - Enable duplicate code identification
  - Enable detection of conflicts

- Ongoing work: User-first documentation
  - Repository for all opensource eBPF programs
  - GUI + ElasticSearch for quick+advanced searches
Roadmap

- Background
- Understanding Code (Registry)
- Extending Code (OPENED)
- Orchestrating @Scale (Apiary)
eBPF Programs are Monoliths

One Off Programs

Complex Codebase(s)

Observability

Network Functions

*Code from a Katran function
Implications of Monolith on Developer Productivity

Developing a new program

Extracting and reusing functionality is non-trivial

Find sub functionality on GitHub

Step 1: Extract lines
Step 2: Identify + Extract Deps
Surprise Step 3: Rewrite for your target hookpoint
The OPENED Vision

- Select eBPF function of interest
- Extract Function
- Transform Function
- Extends/BUILDs on Transformed Function

OPENED Framework
Extract eBPF func as an independently loadable module

- Identify all dependencies of the eBPF function
  - Dependencies: function call graph, Maps & associated structures, header files
  - Extract relevant dependencies while
    - Ensuring correctness and minimizing technical debt

- Caveat: original code must pass verification

```c
/SEC("decap")
int xdpdecap(struct xdp_md* ctx) {
    fn1()
    fn2()
    fn4()
    fn6()
    fn5()
    fn21
    fn3()
}
```
Challenge: Dependency Extraction

Function Dependencies

- Extended codequery tool[1],
  - Recursively identify function call graph
  - Uses cscope and ctags and sqlite internally
- Used TXL source transformation tool to annotate the function definitions
- Python script to extract functions and header files

```c
/* Extracted from 
/root/github/demo_lpc/codequery/katran/decap_kern.c 
startLine: 223 endLine: 247 */
SEC("decap")
int xdpdecap(struct xdp_md* ctx) {
```

```c
extracted.c
... fn3.1() fn3() fn2() fn1() ...
```

```
/SEC("decap")
int xdpdecap(struct xdp_md* ctx) {
```
Challenge: Dependency Extraction

Map Definitions

- eBPF specific method of tracking 
  `bpf_map_update/lookup_elem` while parsing call flow graph
- TXL code transformation tool to annotate maps and other data structures needed
- Python script to extract structures

```c
bpf_map_lookup_elem(&decap_dst,..)
bpf_map_update_elem(lru_map,..);
```

```
struct lru_map{}
struct decap_dst{}
```
Challenge: Dependency Extraction

Multiple declaration of dependencies (both maps & functions)

```
{#funcName,count,[FileName,lineNumber]}

......
increment_quic_cid_version_stats,1,[<dir...>/balancer_kern.c,445]
increment_quic_cid_drop_no_real,1,[<dir...>/balancer_kern.c,460]
process_l3_headers,2,[<dir...>/balancer_kern.c,158],[<dir...>/decap_kern.c,34]
increment_quic_cid_drop_real_0,1,[<dir...>/balancer_kern.c,470]
process_encaped_ipip_pckt,2,[<dir...>/balancer_kern.c,340],[<dir...>/decap_kern.c,85]
parse_udp,1,[<dir...>/pckt_parsing.h,76]
REPORT_PACKET_TOOBIG,2,[<dir...>/introspection.h,32],[<dir...>/introspection.h,40]
......
```
Preserve MACRO Definitions during Extraction

Identify and propagate preprocessor guards into extracted code

**balancer_kern.c**

```c
#ifdef GLOBAL_LRU_LOOKUP

__attribute__((__always_inline__)) static inline bool
reals_have_same_addr(  
    struct real_definition* a,
    struct real_definition* b) {

  ...

}

__attribute__((__always_inline__)) static inline int
perform_global_lru_lookup(  
    struct real_definition** dst,
    struct packet_description* pckt,..){

  ...

}
#endif // GLOBAL_LRU_LOOKUP
```

**extracted.c**

```c
#ifdef GLOBAL_LRU_LOOKUP

/* Extracted from balancer_kern.c startLine: 261 endLine: 277 */

__attribute__((__always_inline__)) static inline bool
reals_have_same_addr(  
    struct real_definition* a,
    struct real_definition* b) {

  ...

}

#endif

#ifdef GLOBAL_LRU_LOOKUP

/* Extracted from balancer_kern.c startLine: 279 endLine: 337 */

__attribute__((__always_inline__)) static inline int
perform_global_lru_lookup(  
    struct real_definition** dst,
    struct packet_description* pckt,..){

  ...

}

#endif // GLOBAL_LRU_LOOKUP
```

Identify and propagate preprocessor guards into extracted code.
Challenge: Minimize Code Debt

- Maintain ordering between definitions and invocations
- Propagate license into newly created c file (with extracted code).
- Identify and copy relevant current directory includes into extraction site.
  - Introduce preprocessor guards in new header files
- Rewrite Makefiles (currently Manual).

```c
#include balancer_const.h
#ifdef BALANCER_CONST_OPF
#include balancer_const.h
#endif
```
**OPENED Vision:** Reduce time to new functionality development

- Automated extraction of relevant code
- Automated transformation of code

**Ongoing work:**
- Create a library of common functions
- Create interface for introspecting into deployed functions

Demo: [https://lpc.events/event/16/contributions/1370/](https://lpc.events/event/16/contributions/1370/)
Roadmap

- Background
- Understanding Code (Registry)
- Extending Code (OPENED)
- Orchestrating @Scale (Apiary)
Kubernetes is to Containers
As
?????? is to eBPFPrograms

Deployed programs = func (Apps, Role)

Infra Dynamics
App-Specific Policies

Unique program configurations

Constantly Deploy/undeploy
DevOps

Service policy

Apiary (eBPF Orchestrator)

Deployment Updates

Lifecycle Management

Infrastructure

Kernel

Server

agent

agent

agent

kernel

Server

Apiary

Kubernetes
Open Problems and Deployment Challenges

- Code Portability
- High level Language Abstractions
- Debugging & Diagnosis
DevTools for Supporting @Scale Adoption of eBPF

eBPF is terraforming modern infrastructures

Ecosystem lacks support for largescale DevOps

Ensemble of tools for lifecycle management

Join Us!
Submit your use cases for programs to be decomposed

https://github.com/eBPFDevSecTools