eBPF в production-условиях

Дмитрий Евдокимов,
Александр Трухин

About us

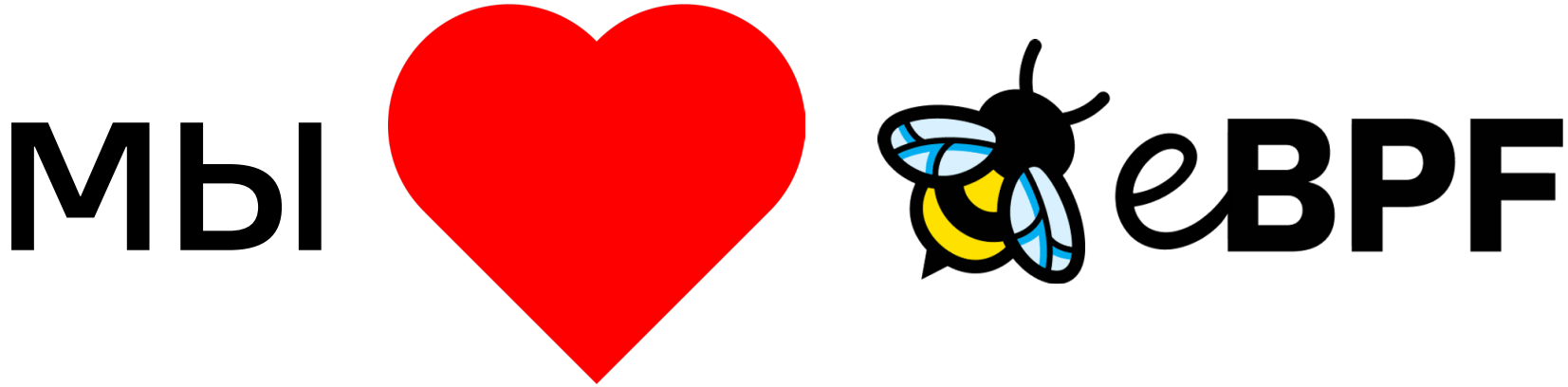
Дмитрий Евдокимов

Founder, CTO Luntry

Александр Трухин

Разработчик, Luntry

Disclaimer



Agenda

- Введение в тему
- Возможности eBPF
- Разработка с eBPF
- Портируемость eBPF кода
- Производительность eBPF кода
- Безопасность eBPF
- Заключение

Введение в тему

eBPF

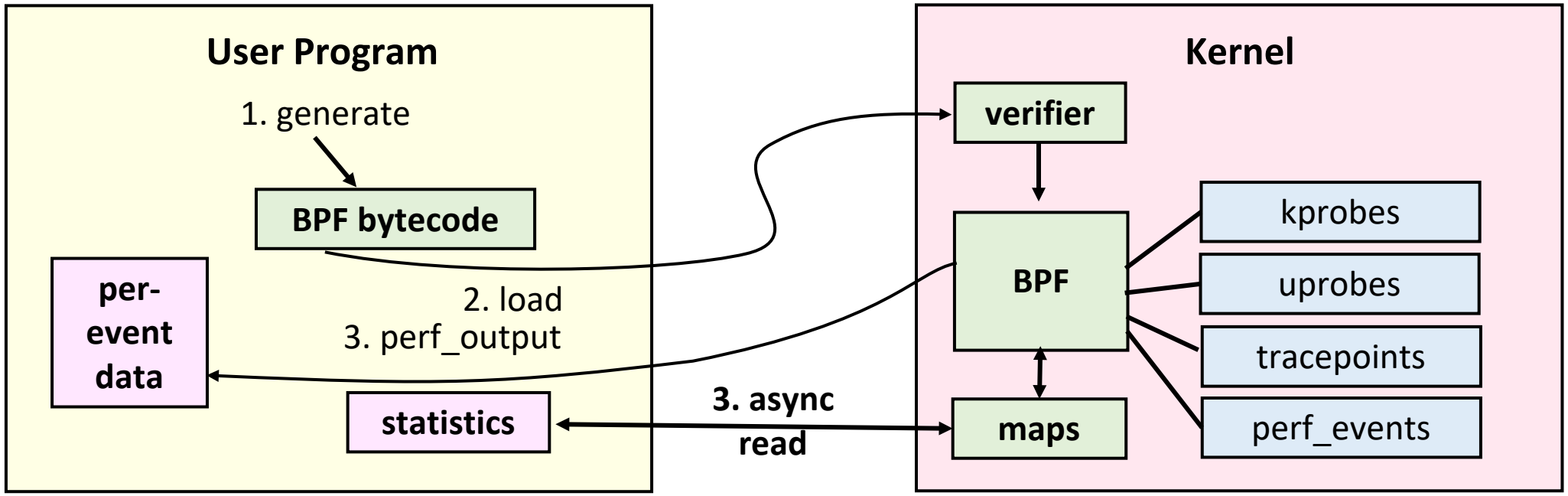
- Развитие BPF
  - extended Berkeley Packet Filter
- Event-driven природа

eBPF Foundation

Founding Members

FACEBOOK Google ISOVALENT

Microsoft NETFLIX



eBPF vs Kernel Modules

	Kernel Module	eBPF программы
Возможности	Безграничные	Ограниченные
Разработка	Все прелести Kernel и Си	Kernel space (Си\rust) + user space (Go\Python\C\Rust)
Портируемость	Зависит от версии ядра	С меньшей зависимостью от версии ядра
Производительность	Высокая	Высокая (JIT специфика) + гарантия выполнения за конечное время
Безопасность (safety)	Ошибка может привести к падению хоста	Необработанные ошибки не предполагаются
Безопасность (security)	Безграничные возможности	Ограниченное влияние на систему

Постоянно в развитии

- Появился в 2013
- eBPF “доступен” с версии 3.15
- Чем новее ядро, тем больше возможностей и меньше ограничений

https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md

Feature	Kernel version
`AF_PACKET` (libpcap/tcpdump, `cls_bpf` classifier, netfilter's `xt_bpf`, team driver's load-balancing mode...)	3.15
Kernel helpers	3.15
`bpf()` syscall	3.18
Tables (*a.k.a.* Maps; details below)	3.18
BPF attached to sockets	3.19
BPF attached to `kprobes`	4.1
`cls_bpf` / `act_bpf` for `tc`	4.1
Tail calls	4.2
Non-root programs on sockets	4.4
Persistent maps and programs (virtual FS)	4.4
`tc`'s `direct-action` (`da`) mode	4.4
`tc`'s `clsact` qdisc	4.5
BPF attached to tracepoints	4.7
Direct packet access	4.7
XDP (see below)	4.8
BPF attached to perf events	4.9
Hardware offload for `tc`'s `cls_bpf`	4.9
Verifier exposure and internal hooks	4.9
BPF attached to cgroups for socket filtering	4.10
Lightweight tunnel encapsulation	4.10
eBPF support for `xt_bpf` module (iptables)	4.10
BPF program tag	4.10
bpftool utility in kernel sources	4.15
BPF attached to cgroups as device controller	4.15
bpf2bpf function calls	4.16
BPF used for monitoring socket RX/TX data	4.17
	4.17
	4.17
	4.18
	4.18
	4.18
	4.18
	4.18
	4.19
	4.20
	5.2
	5.2
	5.2
	5.5
	5.7
	5.8
	5.9
	5.10

eBPF и платформы

Platform →	Linux		MacOSX		Android		FreeBSD		Windows		TockOS (embedded)	
↓ Project	Kernel	User	Kernel	User	Kernel	User	Kernel	User	Kernel	User	Kernel	User
Linux	2014											
uBPF		2015										
rbpf		2017		2017						2018		
Android					2017							
Generic eBPF	2017	2017		2017			2017	2017				
eBPF for Windows									2021			
Tock											2021	

eBPF и архитектуры процессоров

- In-kernel interpreter
- /proc/sys/net/core/bpf_jit_enable

Feature / Architecture	Kernel version	Commit
x86_64	3.16	622582786c9e
ARM64	3.18	e54bcde3d69d
s390	4.1	054623105728
Constant blinding for JIT machines	4.7	4f3446bb809f
PowerPC64	4.8	156d0e290e96
Constant blinding - PowerPC64	4.9	b7b7013cac55
Sparc64	4.12	7a12b5031c6b
MIPS	4.13	f381bf6d82f0
ARM32	4.14	39c13c204bb1
x86_32	4.18	03f5781be2c7
RISC-V RV64G	5.1	2353ecc6f91f
RISC-V RV32G	5.7	5f316b65e99f

Основные мысли раздела

Для разработчиков:

- Портируемость и надежность выше чем у kernel modules
- Kernel space бэкграунд будет плюсом
- Новое ядро это новые возможности

Для пользователей:

- Вы все чаще будете встречать eBPF решения
- Можете спать спокойнее ;)
- Свежее ядро только плюс

Возможности eBPF

Где eBPF может использоваться?

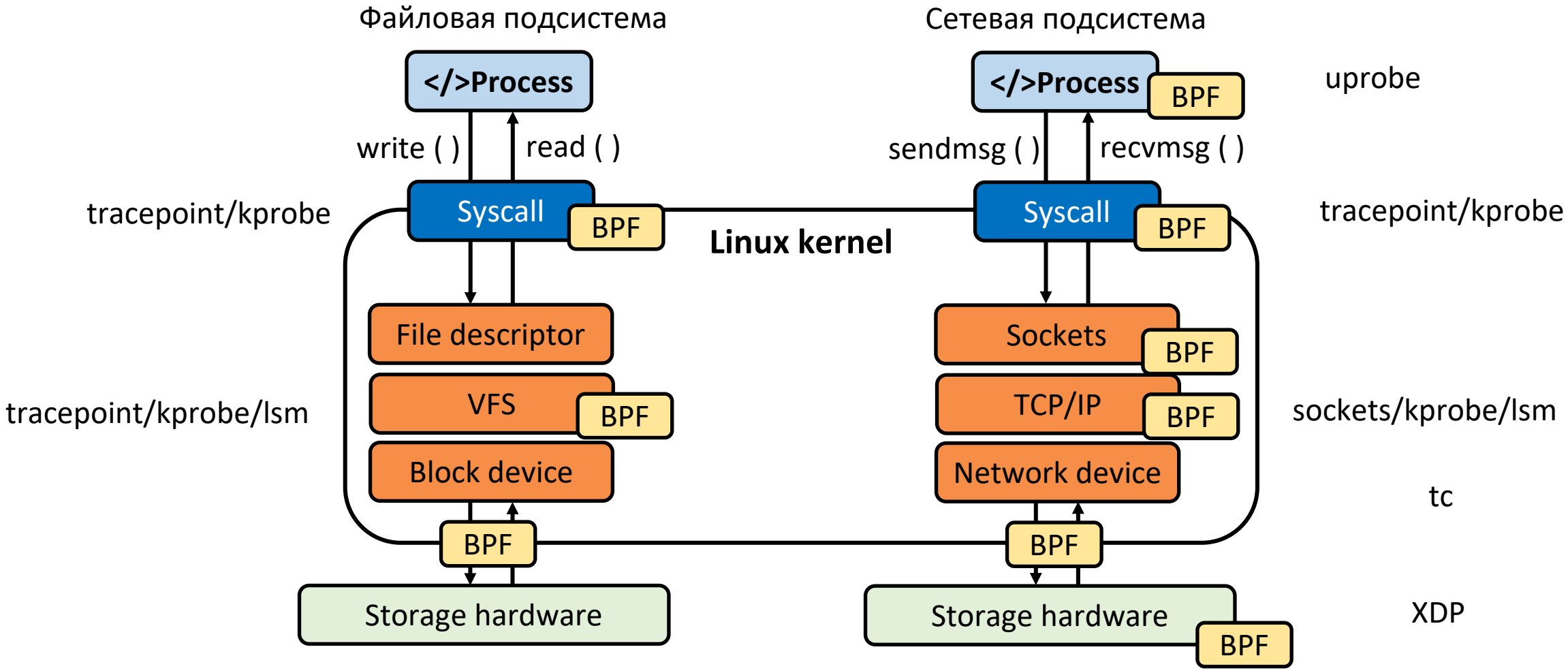
- Networking: Load Balancing, Kubernetes Networking, Service Mesh
  - Cilium - CNI plugin для Kubernetes
  - Katran - layer 4 load balancer
- Observability: Any metrics, logs, traces, continuous profiling,...
  - Pixie - observability решение для Kubernetes
  - Parca - continuous profiling решение
  - SysinternalsEBPF - Linux порт легендарного Sysmon
- Security: Network/Runtime/Data security
  - Falco, Tracee - runtime security решения
  - Inspektor Gadget - набор программ для отладки и интроспекции Kubernetes applications

Типы eBPF-программ

- Около 50 типов программ
  - Смотреть *bpf_prog_type/bpf_attach_type* в bpf.h
  - Разные задачи
  - Разная распространенность
  - От узкоспециализированных до универсальных

Networking	Tracing	LSM
– XDP – tc – sockets – cgroups – …	– tracepoints – raw_tracepoints – kprobes – fentry – uprobes – perf events – …	

До чего может дотянуться eBPF ?



eBPF и containers/sandbox/VM/microVM

- При работе с контейнерами можно определить контекст
- При работе с sandbox/VM/microVM нельзя определить контекст
  - VM Wasmer и подобные
  - Sandbox gVisor
  - Kata Runtime
  - MicroVM Fargate
    - [Fargate] [request]: Provide the ability to use ebpf on fargate instances. #1027

Основные мысли раздела

Для разработчиков:

- Широкая область применения - главное найти ;)
- Плохая документация - много только из примеров и рассылок
- Нужно ориентироваться в типах eBPF программ

Для пользователей:

- Дружит с классическими контейнерами
- Есть решения для network, observability, security

Разработка с eBPF

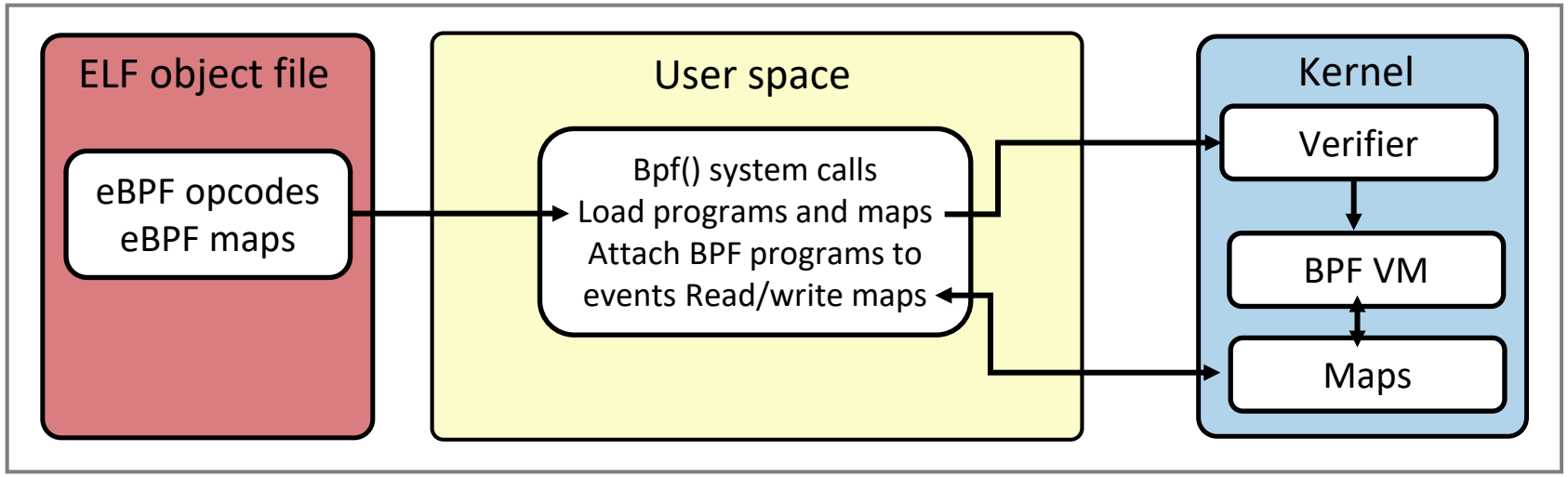
Написание и сборка eBPF-программ

User-space:

- C – libbpf
- Python – bcc
- Go – libbpf-go
- Rust – libbpf-rs

Kernel-space:

- Обычно C
- bcc берет все на себя
- Компилятор llvm/gcc
- На выходе elf “.o” файл



Ограничения eBPF

- Выполнение кода за конечное время
  - нет бесконечных циклов
  - нет I/O
  - код нигде не может зависнуть
- Только ограниченное влияние на систему
  - нельзя исполнить произвольный код из ядра
  - нельзя модифицировать память ядра
- Размер кода
  - linux < 5.2 : 4096 инструкций
  - linux > 5.2: 1000000 инструкций
- Сложность кода (симуляция исполнения кода)
  - linux < 5.2 : 131072 инструкций
  - linux > 5.2: 1000000 инструкций
- Стек 512 байт
- Доступ к некоторым данным возможен только через специализированные kernel helpers
- ...

Основные мысли раздела

Для разработчиков:

- Используйте готовые библиотеки
- eBPF имеет большое количество ограничений
- Чем новее ядро, тем проще с ограничениями
- Не надо и не возможно все реализовать в eBPF

Для пользователей:

- Не мешайте разработчикам ;)

Портируемость eBPF кода

Проблема 1: нестабильность интерфейсов в ядре

1. Сам по себе eBPF-байткод архитектурно-независим и обратно совместим со старыми версиями ядра
2. Проблему составляет работа с нестабильными структурами ядра: чтение и обработка их полей
3. Сигнатуры функций могут меняться, функции могут удаляться в новых версиях ядра

```
struct task_struct {
    ...
    pid_t pid; // offset X
    ...
};
```



```
struct task_struct {
    ...
    long _;
    pid_t pid; // offset X + 8
    ...
};
```

```
struct task_struct task;
...
_READ(task->pid)
```

Развитие решения проблемы

- Стабильные интерфейсы
- Использование Kernel headers
  - CONFIG_IKHEADERS
- Отгадывание offsets
- BTF и CO-RE
  - BTFHub

Стабильные интерфейсы

Linux headers

- Собирать при запуске
  - Нужен компилятор (llvm/gcc)
  - Нужны заголовки ядра
    - обычно доступны по пути (из системного пакета): /lib/modules/$(uname -r)/build/
    - при наличии опции CONFIG_IKHEADERS: /sys/kernel/kheaders.tar.xz
    - могут быть доступны для загрузки из сети
      - известная система (например, Google COS (Container-Optimised OS))
- Предсобранный вариант
  - Или под каждое ядро своя сборка

Отгадывание offsets

florian
@0x0F10

offsets are an important topic when writing eBPF tools. BTF and BPF skeleton can be an answer to this. So it is great to see @kevsecurity did find a different solution with github.com/Sysinternals/S... great work!

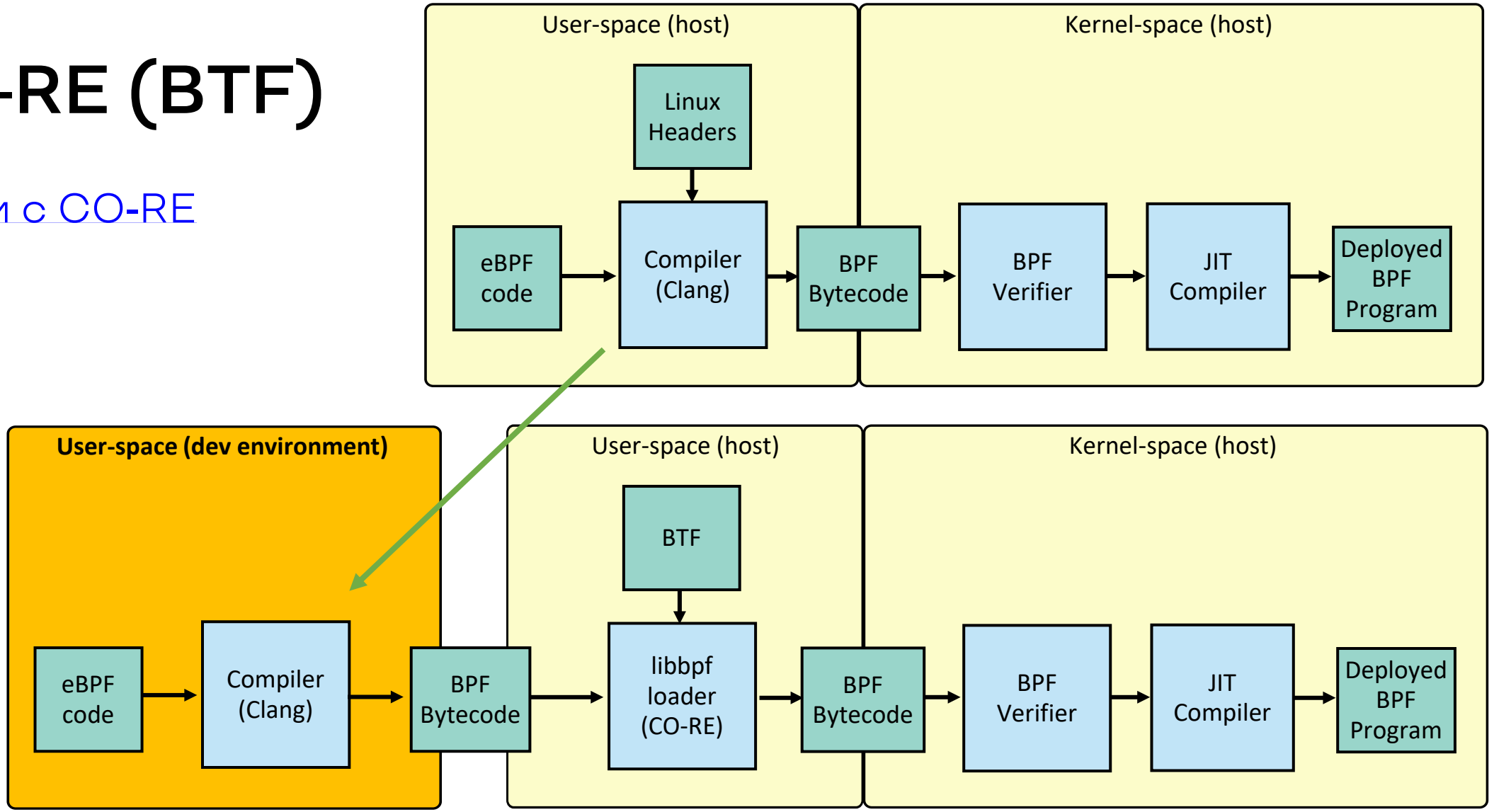
https://github.com/Sysinternals/SysinternalsEBPF/blob/main/discoverOffsets.c#L25

CO-RE (BTF)

- Концепция CO-RE: Compile Once – Run Everywhere
- На 90% состоит из BTF: BPF Type Format
- Проблема разных оффсетов полей структур решается на этапе загрузки с помощью релокаций
- Собирается eBPF код только один раз
- На всех системах используется одна и та же сборка
- BTF встроено в ядро с 5.2
  - CONFIG_DEBUG_INFO_BTF=y
  - bpftool btf dump file /sys/kernel/btf/vmlinux format c

CO-RE (BTF)

Без и с CO-RE



BTFHub

What is BTFhub ?

Unfortunately the BTF format wasn't always available and, because of **missing kernel support**, or because of the lack of userland tools, capable of understanding the BTF format, distributions release(ed) kernels without the embedded BTF information.

That is why BTFhub exists: to provide BTF information for Linux distributions released kernels that don't have embedded BTF information. Instead of recompiling your eBPF code to each existing Linux kernel that does not support BTF information, your code will be relocated - by libbpf - according to available BTF information from the BTFhub files.

Про ОС и managed Kubernetes в облаках

- Облачные провайдеры стараются быть более eBPF дружелюбными
- Поддержка BTF на новых версиях ядра (>= 5.2) как правило имеется
- Появляются Cloud/Kubernetes oriented OS с целенаправленной поддержкой eBPF
  - Пример: Flatcar Container Linux
- RHEL
  - Нужно доставлять kernel header через machine-config
  - Много чего бэкпортят на старые ядра

Проблема 2: прохождение kernel verifier

1. При загрузке kernel verifier проверяет eBPF-код на корректность
2. Разные версии ядра могут иметь разные представления о корректности
3. Разные версии компилятора и разные оптимизации могут вмешиваться в этот процесс

Решение:

- Тестировать
- При необходимости добавлять дополнительные safeguards в код
- Придерживаться требований минимально поддерживаемой вами версии ядра

Основные мысли раздела

Для разработчиков:

- Нужен план “B” если чего-то не будет в новой версии ядра
- Много головной боли со старыми версиями ядра
- Лучше использовать BTF (если не надо поддерживать старые ядра)
- Тестируйте на всех целевых ядрах
- Зафиксируйте версию компилятора

Для пользователей:

- На более новых ядрах меньше возможных проблем
- На более новых ядрах может быть больше возможностей (функционал)

Производительность eBPF кода

Производительность

- Оверхед грубо складывается из:
  - работа eBPF-программ (kernel space)
  - обработка данных (user space)
- eBPF programs are event-driven
  - производительность сильно зависит от нагрузки, количества событий

Оверхед точек входа (https://lwn.net/Articles/748352/):

samples/bpf/test_overhead performance on 1 cpu:

tracepoint	base	kprobe+bpf	tracepoint+bpf	raw_tracepoint+bpf
task_rename	1.1M	769K	947K	1.0M
urandom_read	789K	697K	750K	755K

Пример с неожиданной нагрузкой lsof

```
110  /*
111   * Close enough file descriptors above 2 that library functions will have
112   * open descriptors.
113   *
114   * Make sure stderr, stdout, and stdin are open descriptors.  Open /dev/null
115   * for ones that aren't.  Be terse.
116   *
117   * Make sure umask allows lsof to define its own file permissions.
118   */
119
120          if ((MaxFd = (int) GET_MAX_FD()) < 53)
121              MaxFd = 53;
122
123  #if     defined(HAS_CLOSEFROM)
124          (void) closefrom(3);
125  #else   /* !defined(HAS_CLOSEFROM) */
126  #if     defined(SYS_close_range)
127          if (MaxFd > 3 && syscall(SYS_close_range, 3, MaxFd - 1, 0) == 0)
128              goto closed;
129  #endif
130          for (i = 3; i < MaxFd; i++)
131              (void) close(i);
132  #if     defined(SYS_close_range)
133    closed:
134  #endif
135  #endif  /* !defined(HAS_CLOSEFROM) */
```

В контейнерах
MaxFd >= 10^6

Основные мысли раздела

Для разработчиков:

- Сценарии с высокой нагрузкой могут оказаться неожиданными

Для пользователей:

- По-хорошему производительность приложений нужно замерять отдельно с/без eBPF решения

Безопасность eBPF

Safety и security eBPF

- Safety – достигается/обеспечивается благодаря verifier и ряду функциональных ограничений
- Security – достигается/обеспечивается качеством кода и архитектуры

Уязвимости в подсистеме eBPF

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=BPF

Вредоносный код на eBPF

- Исследования:
  - “With Friends like eBPF, who needs enemies?”
  - “Warping Reality - creating and countering the next generation of Linux rootkits using eBPF”
  - “Evil eBPF”
- Проекты:
  - ebpfkit - rootkit powered by eBPF
  - boopkit - Linux eBPF backdoor over TCP
  - bad-bpf - collection of eBPF programs demonstrating bad behavior
- Вредоносный код ITW:
  - The Bvp47 - Backdoor of US NSA Equation Group
  - BPFDoor - passive network implant for Linux

Про capability для eBPF

Есть большая разница использования под root и без!

```
CAP_BPF (since Linux 5.8)
       Employ privileged BPF operations; see bpf(2) and
       bpf-helpers(7).

       This capability was added in Linux 5.8 to separate out BPF
       functionality from the overloaded CAP_SYS_ADMIN
       capability.
```

https://man7.org/linux/man-pages/man7/capabilities.7.html

Подпись eBPF программ?!

- Сообщество обсуждаем возможность подписи eBPF программ
  - Статья “Toward signed BPF programs”
  - Сложность из-за структуры программы и JIT компиляции
  - Требует перенос user-space BPF loader в kernel или специального формата файла или специализированный loader

* ~~Удалить/запретить bpf(2) syscall~~

** eBPF можно отключить, пересобрав ядро

Основные мысли раздела

Для разработчиков:

- “С большой силой приходит большая ответственность”, Бен Паркер, дядя Питера Паркера

Для пользователей:

- Обновляем ядро
- Контролируем что и как использует eBPF
- Не раздаем права направо и налево - следуем принципу наименьших привилегий

Заключение

Выводы

1. eBPF это очень круто!
2. eBPF не всемогущ!
   a. Много зависит от окружения, в котором будет работать eBPF код
   b. Много зависит от задач, которые вы будете решать с помощью eBPF

Полезные ссылки

1. The Beginner's Guide to eBPF
2. Цикл статей А. Протопопова
3. BPF Library Ecosystem Overview in Go, Rust, Python and C
4. "Beyond the Hype: Cloud Native eBPF"
5. BPF CO-RE (Compile Once – Run Everywhere)
6. "The Cross-Platform Future of eBPF“
7. "What is eBPF?“
8. "Security Observability with eBPF“

Спасибо за внимание!

Email: de@luntry.ru

Twitter: @evdokimovds

Telegram: @Qu3b3c

luntry.ru