
# Livepatching: «точечные» обновления ядра Linux без перезагрузки

Евгений Шатохин
e.shatokhin@yadro.com

# (Up)time =

224

Август 2024 г.:

# 224

потенциально важных CVEs в ядре Linux
(CVSS >= 7.0)

https://lore.kernel.org/linux-cve-announce/

# Live kernel patching (livepatch)

- Модуль ядра Linux (loadable kernel module)
  - Исправленный код соотв. функций ядра Linux (“точечная правка”)
  - Метаданные для подсистемы “livepatching” в Linux
- Применяется в runtime
  - Apply: load module + enable; remove: disable + unload
  - Старый и новый (исправленный) код сосуществуют в памяти.
  - Процессы - по очереди.
  - stop_machine() не нужна (x86-64, PowerPC, s390x).
- /sys/kernel/livepatch
- Документация:

  https://www.kernel.org/doc/html/latest/livepatch/livepatch.html

# Livepatch: demo

Ubuntu 24.04.1 x86_64, kernel 6.8.0-41-generic

Syzkaller: General protection fault in nf_tproxy_laddr4 (netfilter).

```
Mainline: commit 21a673bddc8f, May 13, 2024
Author: Florian Westphal <fw@strlen.de>
  netfilter: tproxy: bail out if IP has been disabled on the devices

net/ipv4/netfilter/nf_tproxy_ipv4.c:
@@ -58,6 +58,8 @@ __be32 nf_tproxy_laddr4(struct sk_buff *skb, __be32
user_laddr, __be32 daddr)

        laddr = 0;
        indev = __in_dev_get_rcu(skb->dev);
+       if (!indev)
+               return daddr;
        [...]
```

https://syzkaller.appspot.com/bug?extid=b94a6818504ea90d7661

# Demo

# Rebootless patching

- Развитие
  - 2008 - Ksplice
  - 2014 - KPatch (RedHat), kGraft (SUSE); KernelCare / TuxCare
  - 2015 - Livepatch в ядре Linux 4.0 (RedHat + SUSE)
  - 2019 - production-ready Livepatch в ядре Linux 5.1
- Production
  - RedHat, SUSE, Canonical, ... - Livepatch
  - Oracle - Ksplice
  - CloudLinux - TuxCare / KernelCare

# Патчим то, что важно

- RedHat:

  *«Linux kernel live patching is a way to apply critical and important security patches [...]»*

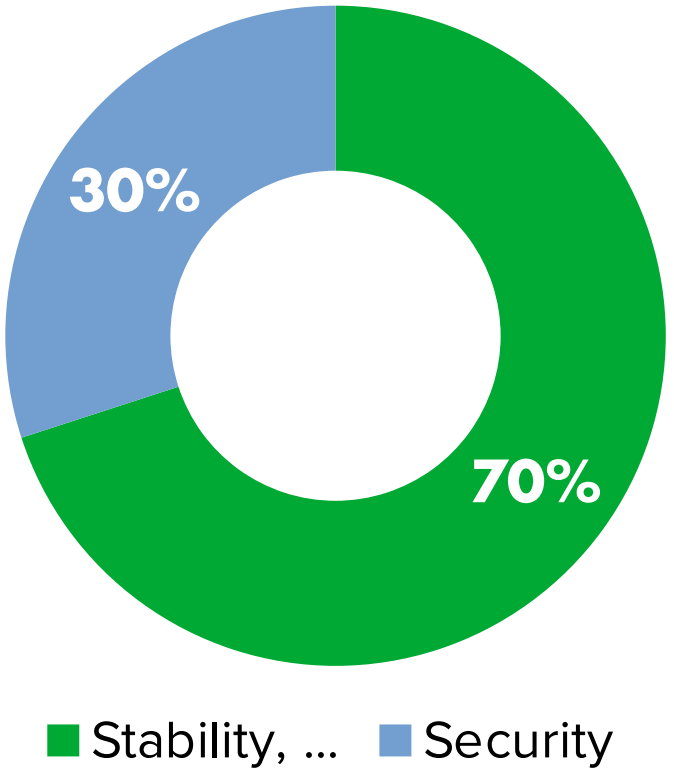
- SUSE:

  *«Live Patching [...] can be used to perform critical security updates and/or serious bug fixes»*

- Canonical:

  *«Canonical Livepatch patches high and critical linux kernel vulnerabilities [...] »*

На практике:



# Ftrace, «function tracing»

- Сбор данных о работе ядра Linux / Android
  - Static tracepoints
  - Dynamic tracing - аргументы и возвр. значения функций, latencies и т.д.
- Расширение функциональности ядра Android
  - Vendor hooks: static tracepoints + vendor modules
- Платформа для других инструментов
  - eBPF, kprobes-on-ftrace, ...
- «Перенаправление» (redirection) функций

https://www.kernel.org/doc/html/latest/trace/ftrace.html
https://www.kernel.org/doc/html/latest/trace/ftrace-uses.html

# Dynamic ftrace, x86-64

nf_tproxy_laddr4(), Ftrace disabled:

```
0f 1f 44 00 00 nopl   0x0(%rax,%rax,1)
55             push   %rbp
89 f0          mov    %esi,%eax
48 89 e5       mov    %rsp,%rbp
[...]
```

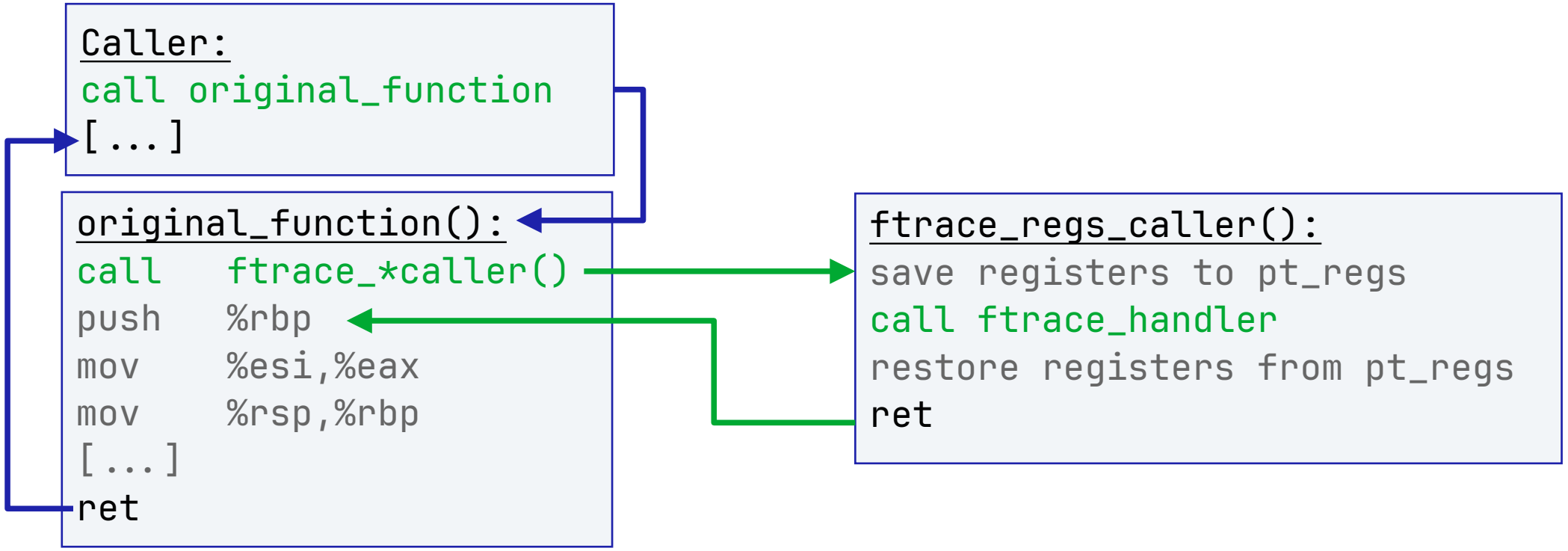
nf_tproxy_laddr4(), Ftrace enabled:

```
e8 9b 9c 00 00 call   ftrace_*caller()
55             push   %rbp
89 f0          mov    %esi,%eax
48 89 e5       mov    %rsp,%rbp
[...]
```

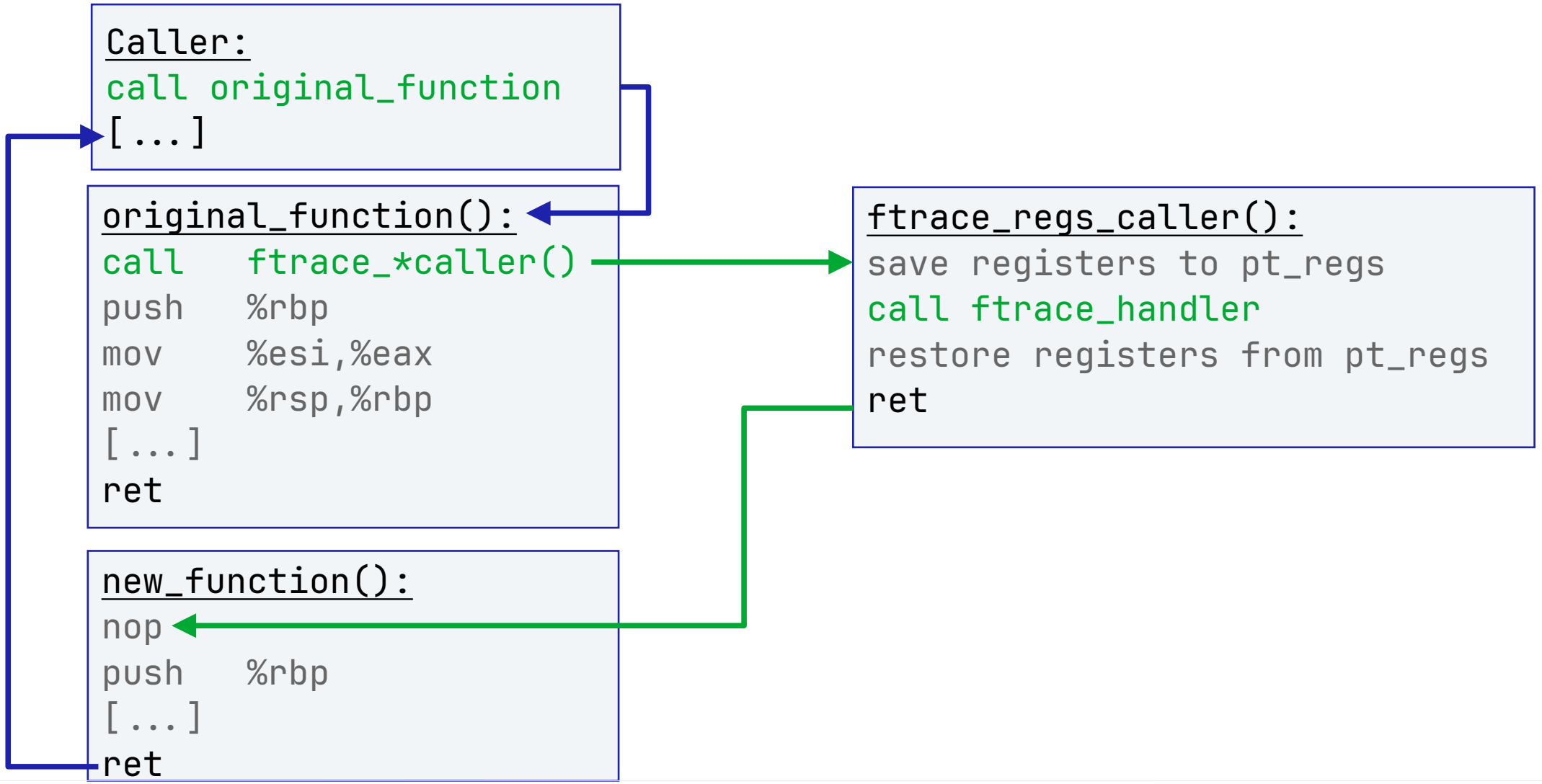
ftrace_*caller(), per-function:

```
save registers to pt_regs

call ftrace_handler_func
# может менять регистры в
# pt_regs и стек

restore registers from pt_regs
ret
```

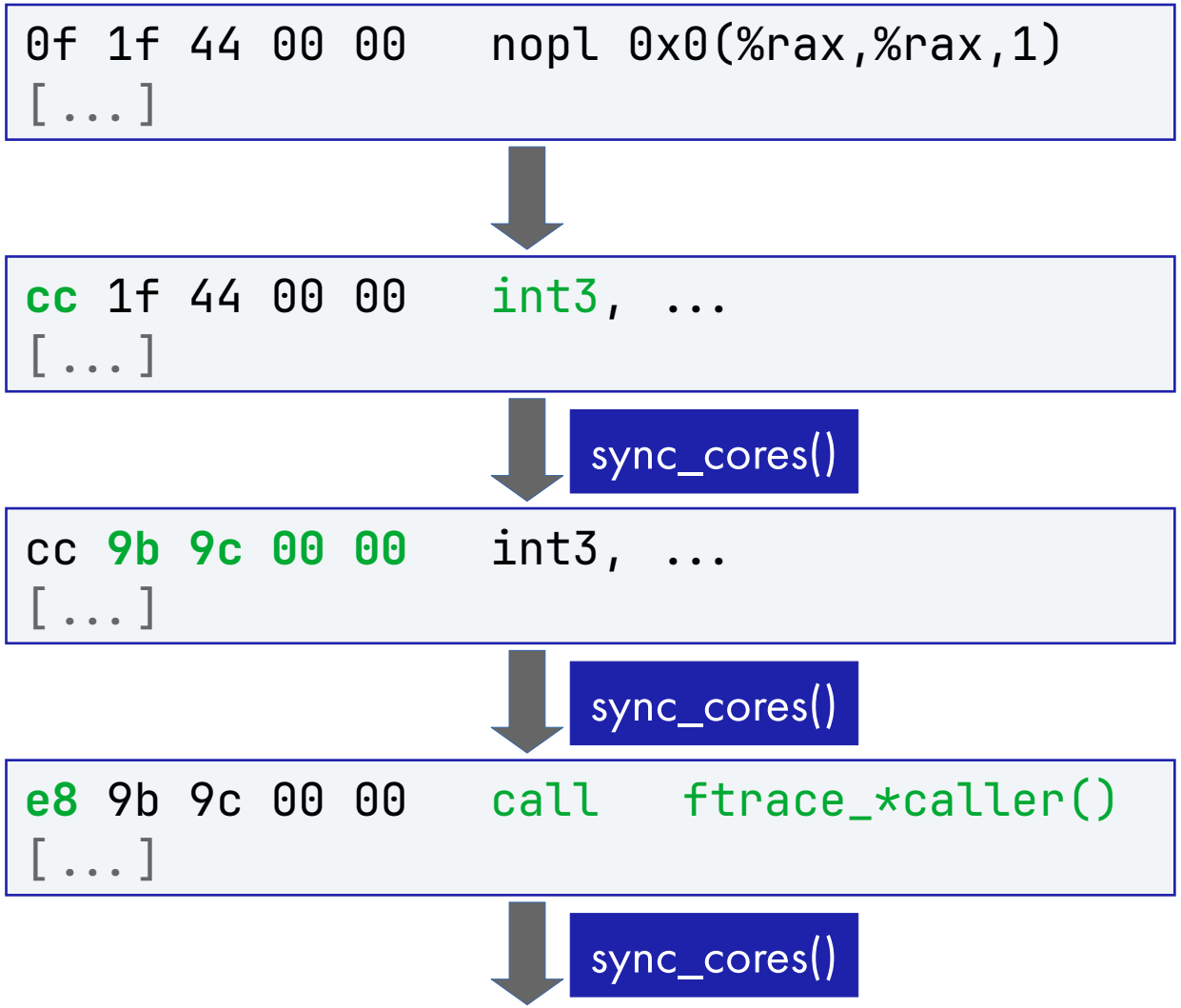
# Ftrace: перенаправление функций



# Ftrace: перенаправление функций



# Dynamic ftrace, x86-64



# Dynamic ftrace, RISC-V

```
13 00 00 00      nop
13 00 00 00      nop
01 11    addi    sp,sp,-32
06 ec    sd      ra,24(sp)
[...]
```


```
97 62 d0 ff      auipc   t0,0xffd06
e7 82 e2 80      jalr    t0,-2034(t0)
                 # <ftrace_caller>
01 11    addi    sp,sp,-32
06 ec    sd      ra,24(sp)
[...]
```

# Dynamic ftrace, RISC-V

```
13 00 00 00      nop
13 00 00 00      nop
01 11    addi    sp,sp,-32
06 ec    sd      ra,24(sp)
[...]
```

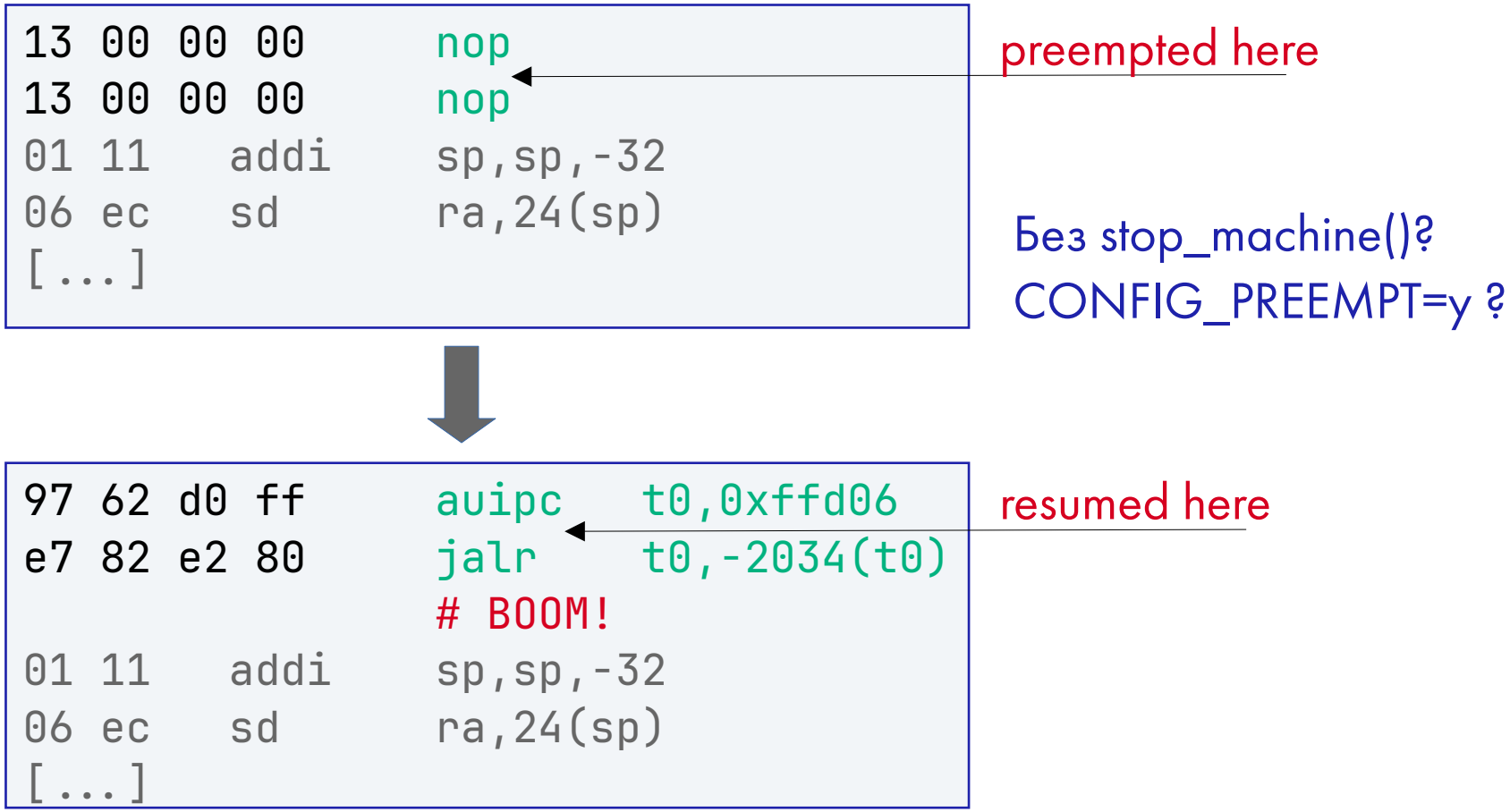
stop_machine()



Дорого!

```
97 62 d0 ff      auipc   t0,0xffd06
e7 82 e2 80      jalr    t0,-2034(t0)
                 # <ftrace_caller>
01 11    addi    sp,sp,-32
06 ec    sd      ra,24(sp)
[...]
```

# Dynamic ftrace, RISC-V



# Livepatch: consistency model

- Важно:
  - Не меняем функции во время их работы
  - Несколько функций - или старый код, или новый, но не смесь
- Consistency model - кто и когда увидит новые функции
  - Per-thread / per-task
  - Thread спит - проверяем stack trace и переключаем.

    *Нужно: reliable stack traces!*
  - Thread выходит в userspace - переключаем.

    *Долго не выходит - signal. 15 с - сигнал - 15 с.*
  - Thread никогда не выходит в userspace - ?

# Поддержка Livepatch, Linux kernel 6.11

| | x86-64 | ppc64le, s390x | AArch64 | RISC-V (RV64G) |
|---|---|---|---|---|
| Dynamic ftrace with function redirection | | | | |
| No stop_machine() | | | | |
| Safe w.r.t. full preemption | | ? | ? | |
| Reliable stack traces | | | | |
| Simple livepatches | | | | |
| Build tools: kpatch-build | | | | |
| New build tools: klp-build | | | | |

Работает | Есть патчи | Не реализовано

# Livepatching - сложно, но не запредельно

- Навыки:
  - Хорошее знание языка C
  - Умение разбираться в коде ядра Linux
  - Знание типовых «подводных камней» в Livepatching
- Open-source инструменты: kpatch tools (RedHat, ...)
  - Сборка livepatch-модулей:
    - kpatch-build (stable),
    - klp-build (in development)
  - Работа с livepatch'ами в runtime - kpatch script

    https://github.com/dynup/kpatch/

# Livepatching

# Анализ и доработка source patch'а

## Обязательно анализируем source patch!

- Что именно делает source патч?
- Возможно ли применить изменения в runtime?
  - Какие функции меняются? Как они связаны?
  - Когда вызываются? __init?
  - Часто ли вызываются? Долго ли работают?
  - Меняет ли патч структуры данных или семантику?
  - ...

Полезно:

https://github.com/dynup/kpatch/blob/master/doc/patch-author-guide.md

# Анализ и доработка source patch'а

```
struct some_struct {
    ...
    struct kref ref;
};
...
struct some_struct *some_obj;
...
some_obj = kzalloc(sizeof(*some_obj), ...);
...
kref_init(&some_obj→ref);
```

- kref_init:
  - refcount = 1
- kref_get(&ref)
  - refcount = refcount + 1
- kref_put(&ref, delete_some_obj)
  - refcount = refcount - 1
  - refcount == 0? delete_some_obj()

# Анализ и доработка source patch'а

```
struct some_struct {
    ...
    struct kref ref;
};
...
struct some_struct *some_obj;
...
some_obj = kzalloc(sizeof(*some_obj), ...);
...
kref_init(&some_obj→ref);
```

- kref_init:
  - refcount = 1
- kref_get(&ref)
  - refcount = refcount + 1
- kref_put(&ref, delete_some_obj)
  - refcount = refcount - 1
  - refcount == 0? delete_some_obj()

```
static int start_activity(...)
{
    ... // use some_obj
}

static int stop_activity(...)
{
    ... // stop using some_obj
}

int handle_command(int cmd, ...)
{
    ...
    if (cmd == START_ACTIVITY) {
        err = start_activity(..., some_obj);
    } else if (cmd == STOP_ACTIVITY) {
        err = stop_activity(..., some_obj);
    }
    ...
}
// Между start_activity() и stop_activity()
// проходит время.
```

# Анализ и доработка source patch'а

```
static int start_activity(...)
{
    ... // use some_obj
}

static int stop_activity(...)
{
    ... // stop using some_obj
}
```

```
static int patched_start_activity(...)
{
    ...
    kref_get(&some_obj->ref);
    // use some_obj
    ...
}

static int patched_stop_activity(...)
{
    ...
    // stop using some_obj
    kref_put(&some_obj->ref, delete_some_obj);
    ...
}
```

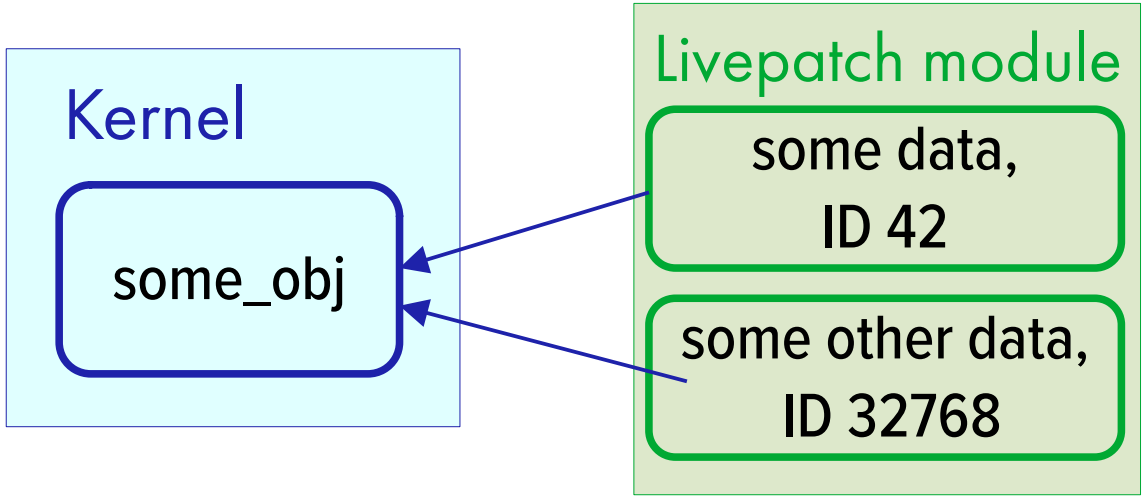
# Анализ и доработка source patch'а

```
static int start_activity(...)
{
    ... // use some_obj
}

static int stop_activity(...)
{
    ... // stop using some_obj
}
```

```
static int patched_start_activity(...)
{
    ...
    kref_get(&some_obj->ref);
    // use some_obj
    ...
}

static int patched_stop_activity(...)
{
    ...
    // stop using some_obj
    kref_put(&some_obj->ref, delete_some_obj);
    ...
}
```

# Анализ и доработка source patch'а

## «Shadow variables»




- `klp_shadow_alloc(&obj, id, …)`
- `klp_shadow_get(&obj, id, …)`
- `klp_shadow_get_or_alloc(&obj, id, …)`
- `klp_shadow_free(&obj, id, …)`
- …

# Анализ и доработка source patch'а

```
static int patched_start_activity(...)
{
   void *patch_data;
   ...
   patch_data = klp_shadow_get_or_alloc(some_obj, 42 /* ID */, ...);
   if (patch_data)
        kref_get(&some_obj→ref);
   // use some_obj
}

static int patched_stop_activity(...)
{
   void *patch_data;
   ...
   // stop using some_obj
   patch_data = klp_shadow_get(some_obj, 42 /* ID */);
   if (patch_data) { // работаем по-новому только в этом случае
      kref_put(&some_obj→kref, delete_some_obj);
   }
   ...
}
// Free - в KPATCH_POST_UNPATCH_CALLBACK() или в delete_some_obj().
```

# Анализ и доработка source patch'а

## Функции без поддержки Ftrace (lib/, notrace, ...)

```
// lib/idr.c:
int idr_alloc_u32(...)
{
    ...
}

// net/9p/client.c
static struct p9_fid *p9_fid_create(...)
{
    ...
    ret = idr_alloc_u32(…);
    ...
}
```

```
// lib/idr.c:
int idr_alloc_u32(...)
{
    ...
}

int patched_idr_alloc_u32(...)
{
    ...
}

// net/9p/client.c
static struct p9_fid *p9_fid_create(...)
{
    ...
    ret = patched_idr_alloc_u32(…);
    ...
}
```

# Анализ и доработка source patch'а

## Не меняем ли лишнего?

```
kpatch-build:
Extracting new and modified ELF sections
nf_tproxy_ipv4.o: changed function: nf_tproxy_laddr4
Patched objects: net/ipv4/netfilter/nf_tproxy_ipv4.ko
```

- inlined функции
- compiler optimizations (*.isra*, *.constprop*, ...)
- __LINE__
- ...

Полезно:

https://github.com/dynup/kpatch/blob/master/doc/patch-author-guide.md

# Livepatch: сделай сам

Для сборки:

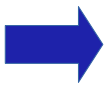
- kpatch tools (kpatch-build)
- Source patch file
- Kernel source tree
- .config
- vmlinux с debug info

```
$ kpatch-build \
  -s ./linux-source-6.8.0 \
  -c ./config-6.8.0-41-generic \
  -v ./vmlinux-6.8.0-41-generic \
   ./netfilter-tproxy-fix.patch
...
Building original source
Building patched source
Extracting new and modified ELF sections
...
Building patch module:
  livepatch-netfilter-tproxy-fix.ko
SUCCESS
```

# Cumulative patches & atomic updates

- "Накопительные"(cumulative) livepatch'и

fix-CVE-2023-6200.patch
fix-CVE-2024-43888.patch
fix-BUG-12382.patch
fix-BUG-32784.patch



fixes-v1.0.patch

livepatch-fixes-v1.0.ko

- Upgrade v1.0 => v2.0 (atomic replacement)

```
# kpatch load livepatch-fixes-v1.0.ko
  ...
# kpatch load livepatch-fixes-v2.0.ko // (replaces and disables v1.0)
# kpatch unload livepatch-fixes-v1.0
```

- Downgrade v2.0 => v1.0 (unload+load)

```
# kpatch unload livepatch-fixes-v1.0
# kpatch load livepatch-fixes-v2.0.ko
```

# Livepatch: тестирование, выпуск, поддержка

- Тестирование
  - Надо!
  - Для каждой версии ядра Linux
  - Семантика патча + штатные проверки
  - Применение / отключение / update / downgrade (!), в т.ч. под нагрузкой
- Постадийный выпуск + мониторинг
  - Подозрительные сбои в системе
  - Ошибки при применении патча
- Поддержка после выпуска

# Livepatching

# Спасибо!
# Вопросы?

# Запасные слайды

# Анализ и доработка source patch'а

```
linux-stable, 4.9.y, CVE-2020-14305
commit 396ba2fc4f27ef6c44bbc0098bfddf4da76dc4c9
Author: Vasily Averin <vasily.averin@linux.dev>
Date:   Tue Jun 9 10:53:22 2020 +0300

    netfilter: nf_conntrack_h323: lost .data_len definition for Q.931/ipv6

--- a/net/netfilter/nf_conntrack_h323_main.c
+++ b/net/netfilter/nf_conntrack_h323_main.c
@@ -1225,6 +1225,7 @@ static struct nf_conntrack_helper
nf_conntrack_helper_q931[] __read_mostly = {
        {
                .name                   = "Q.931",
                .me                     = THIS_MODULE,
+               .data_len               = sizeof(struct nf_ct_h323_master),
                .tuple.src.l3num        = AF_INET6,
                .tuple.src.u.tcp.port       = cpu_to_be16(Q931_PORT),
                .tuple.dst.protonum = IPPROTO_TCP,
```

# Анализ и доработка source patch'a

## Инициализация данных

Commit 54a20552e1ea,
KVM: x86: work around infinite loop in microcode when #AC is delivered»

```
static int (*const svm_exit_handlers[])(struct vcpu_svm *svm) = {
        ...
        [SVM_EXIT_EXCP_BASE + MC_VECTOR]        = mc_interception,
+       [SVM_EXIT_EXCP_BASE + AC_VECTOR]        = ac_interception,
        ...
```

```
@@ -3580,6 +3580,9 @@ static int handle_exit(struct kvm_vcpu *vcpu)
                return 1;
        }

+       if (exit_code == SVM_EXIT_EXCP_BASE + AC_VECTOR)
+               return ac_interception(svm);
+
        return svm_exit_handlers[exit_code](svm);
 }
```

# Работает - не трогай?

```
linux-stable, 4.9.y
commit 396ba2fc4f27ef6c44bbc0098bfddf4da76dc4c9
Author: Vasily Averin <vasily.averin@linux.dev>
Date:   Tue Jun 9 10:53:22 2020 +0300

    netfilter: nf_conntrack_h323: lost .data_len definition for Q.931/ipv6
```

CVE-2020-14305, remote denial of service (RDoS)

CVSS score: 8.1 (High)

«This flaw allows an unauthenticated remote user to crash the system, causing a denial of service.»

(https://nvd.nist.gov/vuln/detail/CVE-2020-14305)