How STACKLEAK improves Linux kernel security

Alexander Popov

Positive Technologies

2017

About Me

- Alexander Popov
- Linux kernel developer
- Security researcher at POSITIVE TECHNOLOGIES

Agenda

- Mission of the Kernel Self Protection Project
- STACKLEAK overview, credit to grsecurity/PaX
- My goal, tactics and the current state
- STACKLEAK as a security feature:
  - Affected kernel vulnerabilities
  - Protection mechanisms
  - Performance penalty
- STACKLEAK inner workings:
  - The asm code erasing the kernel stack
  - The GCC plugin for compile-time instrumentation

Kernel Self Protection Project

- Security is beyond fixing bugs
- Kernel has to fail safely, in addition to running safely
- Goal: eliminate bug classes and methods of exploitation
- Links:
  - KSPP wiki:
    http://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project
  - KSPP overview by Kees Cook:
    https://outflux.net/slides/2017/lss/kspp.pdf

STACKLEAK Overview

- Awesome Linux kernel security feature
- Developed by PaX Team (kudos!)
- `PAX_MEMORY_STACKLEAK` in grsecurity/PaX patch
- grsecurity/PaX patch is now private
- The last public version is for 4.9 kernel (April 2017)

My Goal

Bring STACKLEAK into the Linux kernel mainline

Thanks to Positive Technologies for allowing me to spend part of my working time on it!

My Tactics

- Extract STACKLEAK from grsecurity/PaX patch

```
wc -l ../grsecurity-3.1-4.9.24-201704252333.patch
225976 ../grsecurity-3.1-4.9.24-201704252333.patch
```

- Carefully learn it bit by bit
- Send to LKML, get feedback, improve, repeat

My Tactics

Generally resemble this:

Credit: @EatSleepPwnRpt

Current State

Patch series v5 (22 Oct 2017) for x86_64 and x86_32

http://www.openwall.com/lists/kernel-hardening/2017/10/22/1

21 files changed, 978 insertions(+), 12 deletions(-)

Patches:

1. x86/entry: Add STACKLEAK erasing the kernel stack at the end of syscalls
2. gcc-plugins: Add STACKLEAK plugin for tracking the kernel stack
3. lkdtm: Add test for STACKLEAK (developed together with Tycho Andersen)
4. fs/proc: Show STACKLEAK metrics in the /proc file system
5. doc: self-protection: Add information about STACKLEAK feature

You are welcome to join the review!

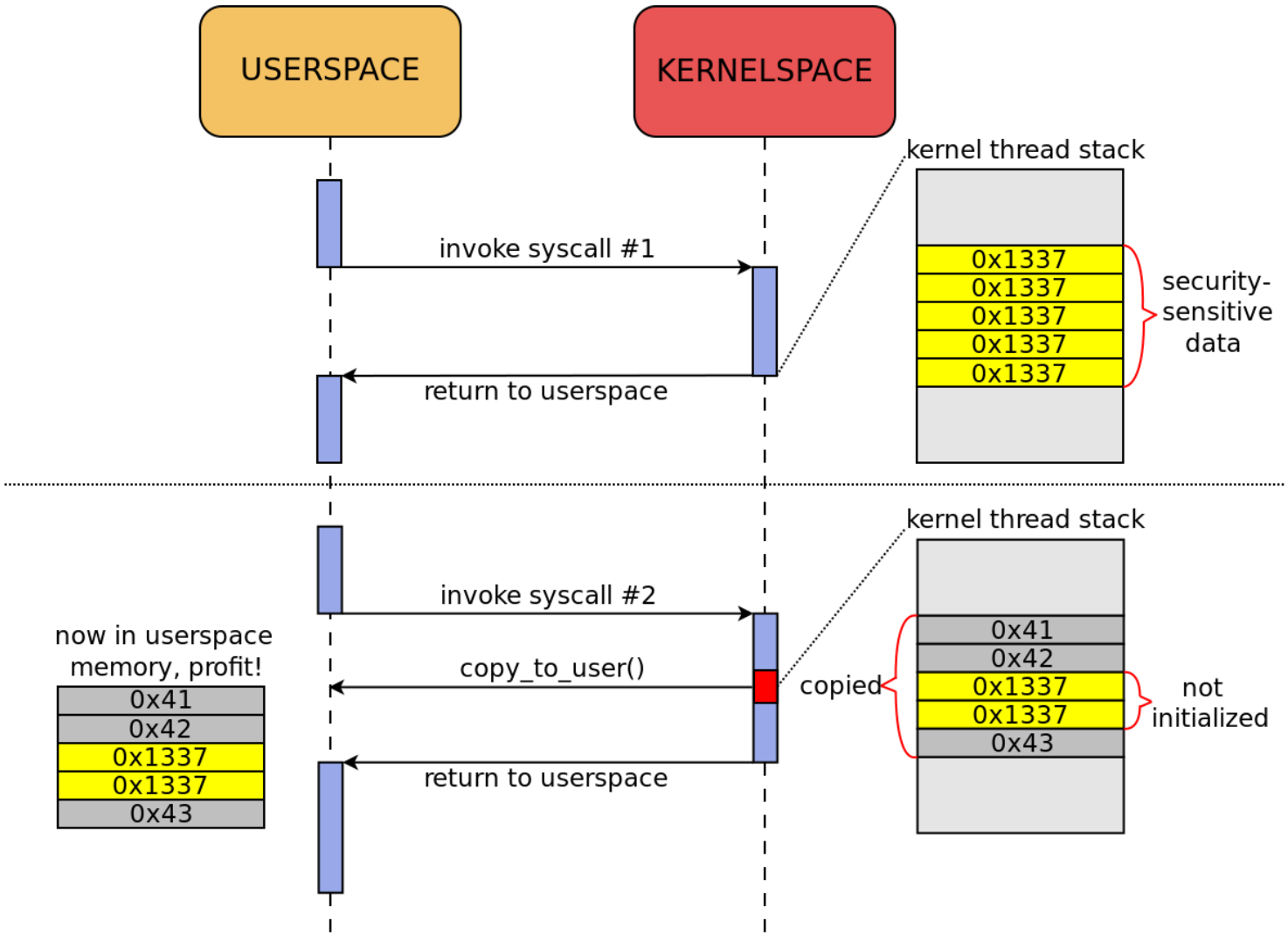
STACKLEAK Security Features

Now about STACKLEAK security features

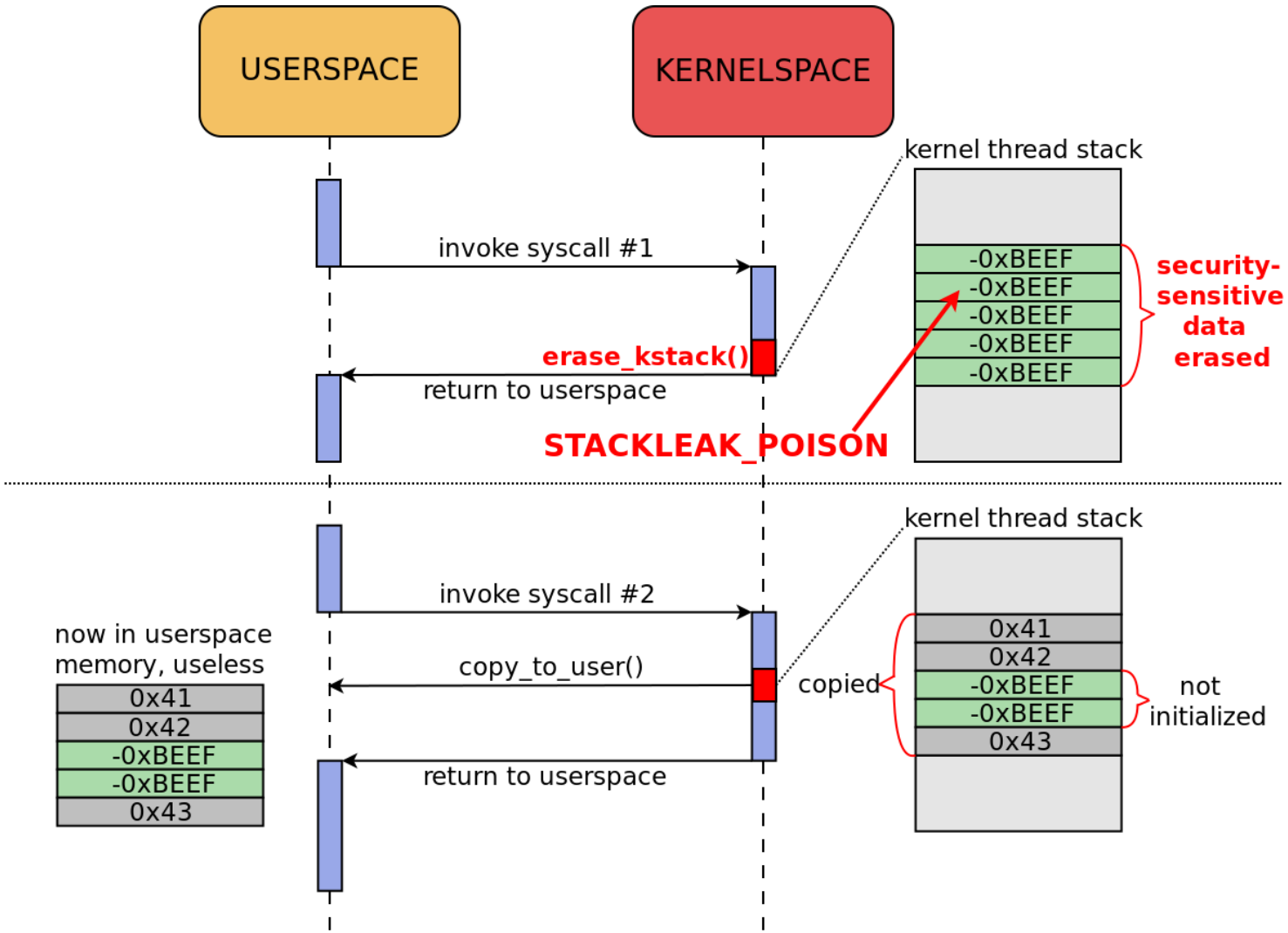
STACKLEAK Security Features (1)

- Erases the kernel stack at the end of syscalls
- Reduces the information that can be revealed through some* kernel stack leak bugs

Kernel Stack Leak Bug Example



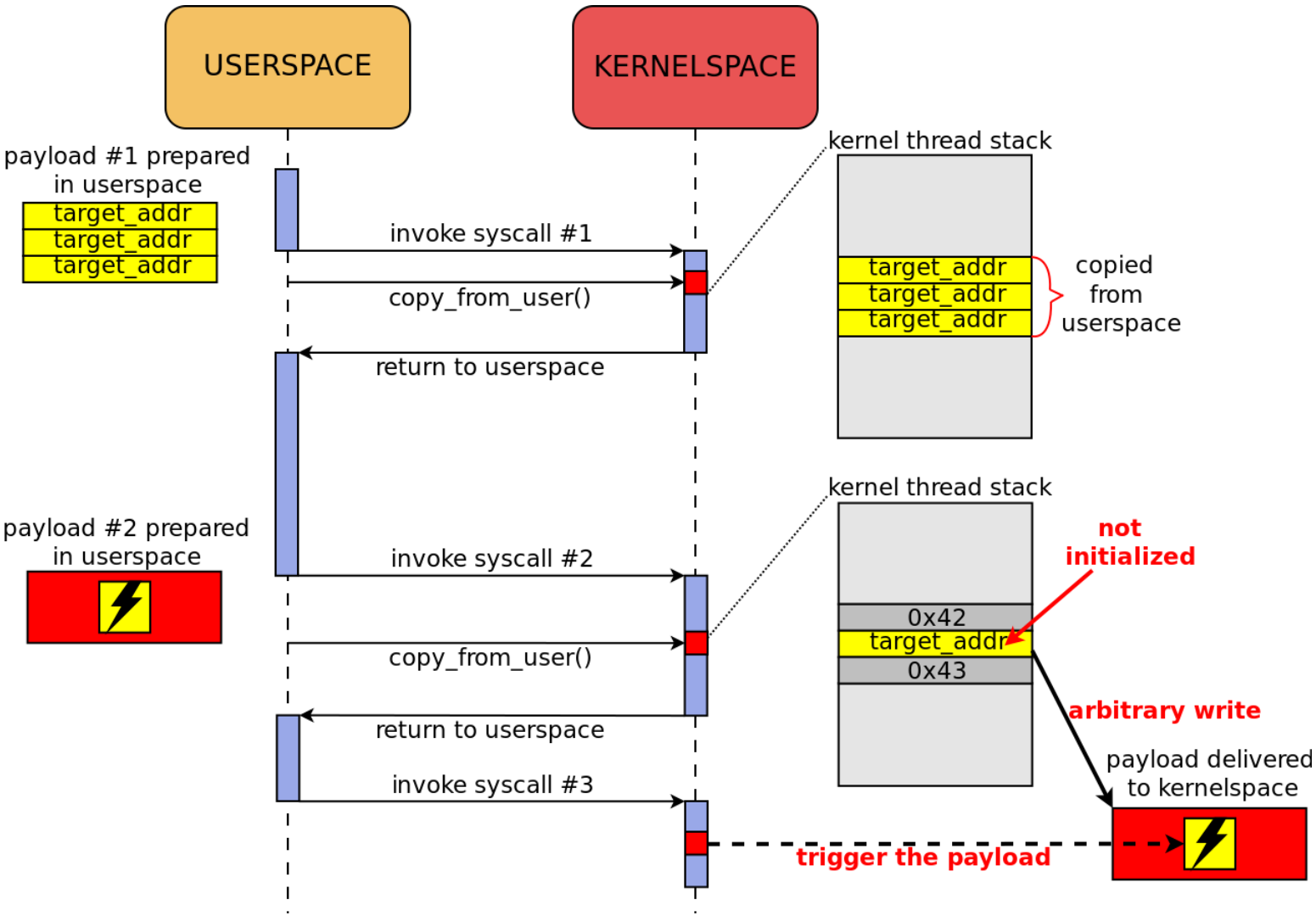
STACKLEAK Mitigation of Such Bugs



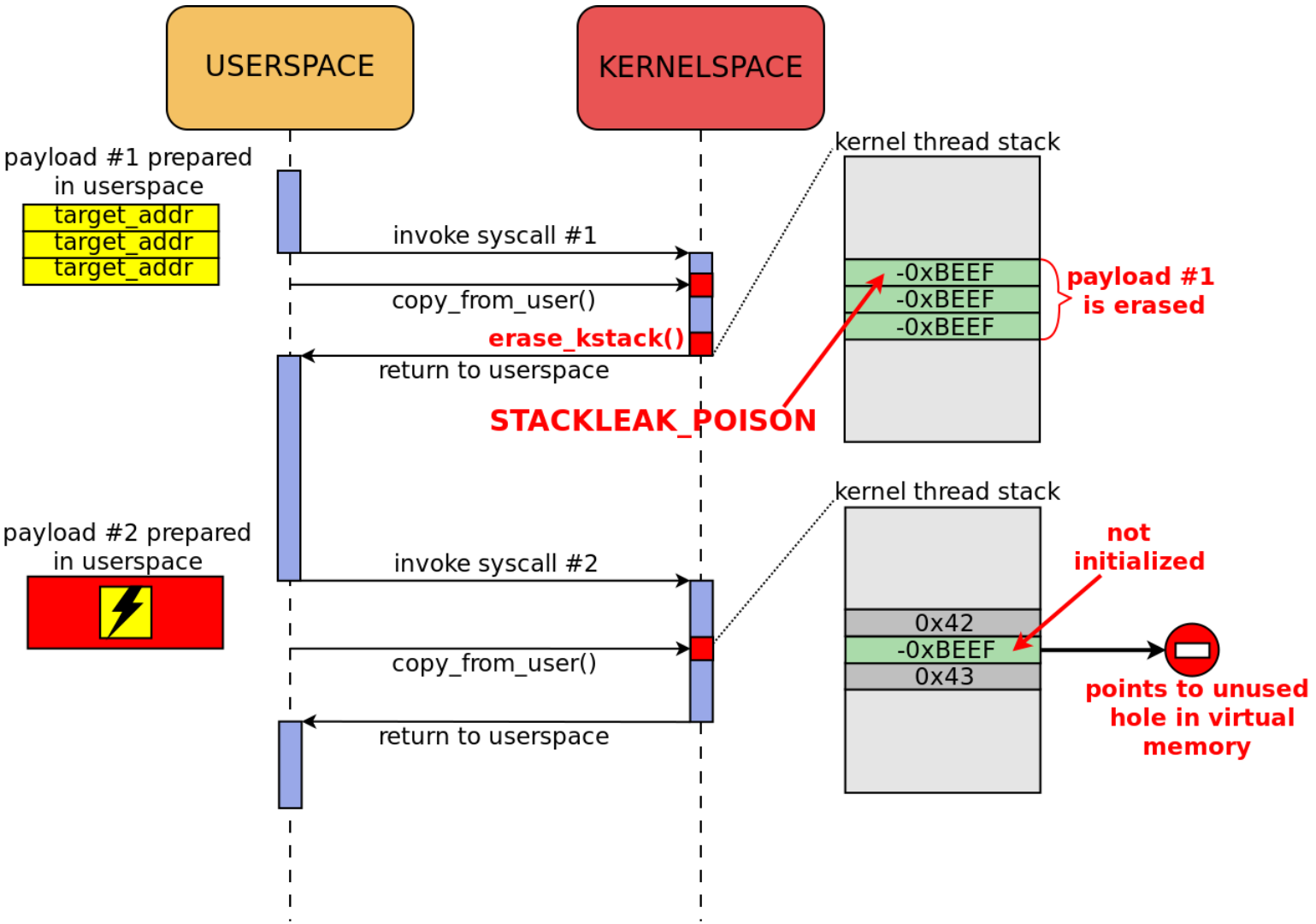
STACKLEAK Security Features (2)

- Blocks some* uninitialized kernel stack variable attacks
- Nice example: CVE-2010-2963 exploitation
- See cool write-up by Kees Cook:
  https://outflux.net/blog/archives/2010/10/19/cve-2010-2963-v4l-compat-exploit/

Uninitialized Stack Variable Attack



Mitigation of Uninitialized Stack Variable Attacks



Important Limitation

* STACKLEAK doesn't help against such attacks during a single syscall

STACKLEAK Security Features (3)

Adds runtime detection of kernel stack depth overflow

Interrelation of Security Mechanisms

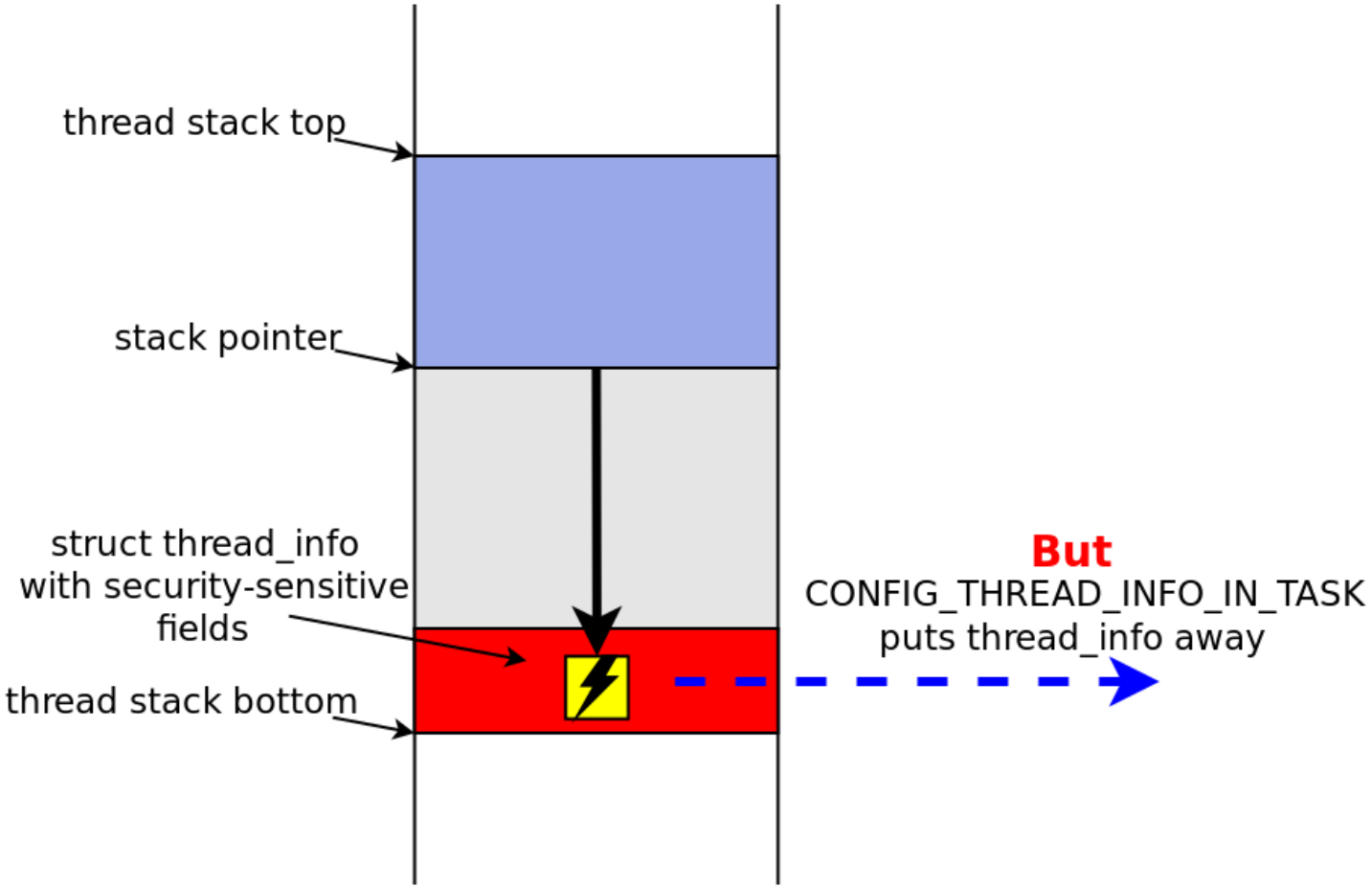
In mainline kernel STACKLEAK would be effective against kernel stack depth overflow only **in combination** with:

- `CONFIG_THREAD_INFO_IN_TASK`
- `CONFIG_VMAP_STACK` (kudos to Andy Lutomirski)

Viktor Vasnetsov, Bogatyrs (1898)

Kernel Stack Depth Overflow (1)

See "The Stack is Back" by Jon Oberheide (2012)

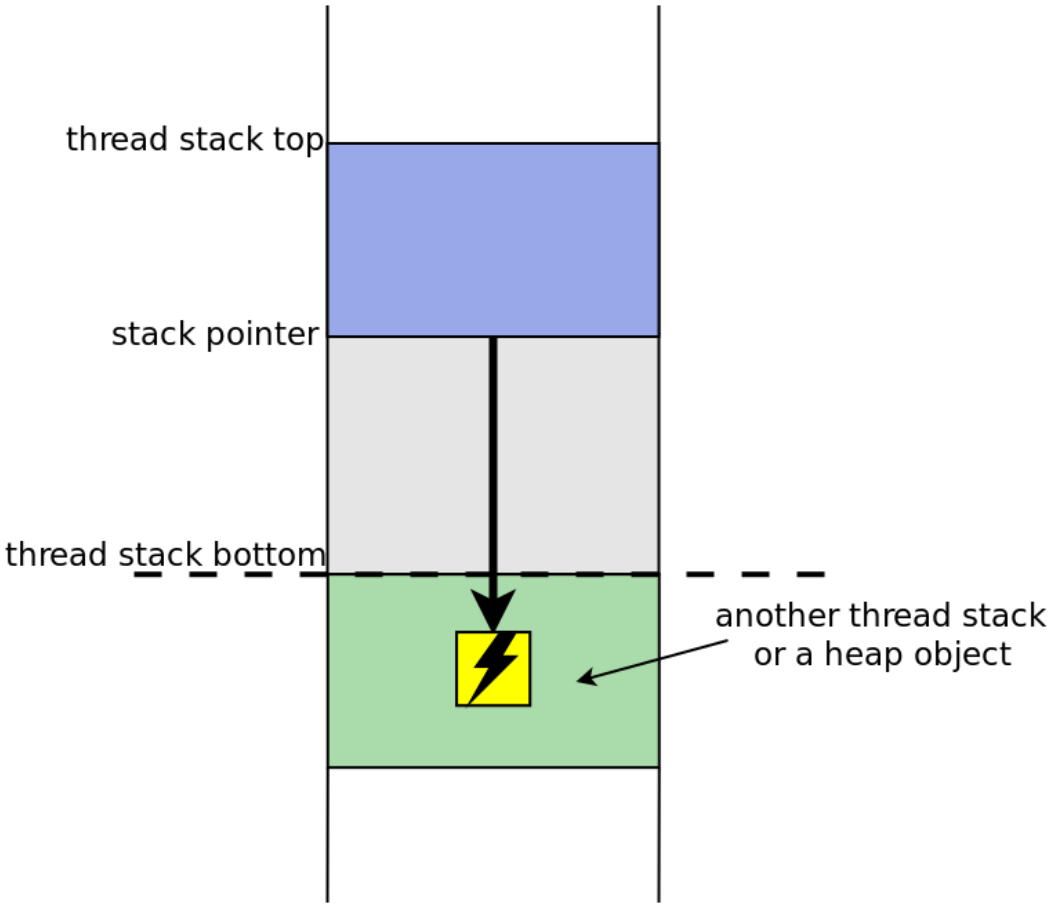


Kernel Stack Depth Overflow Strikes Back

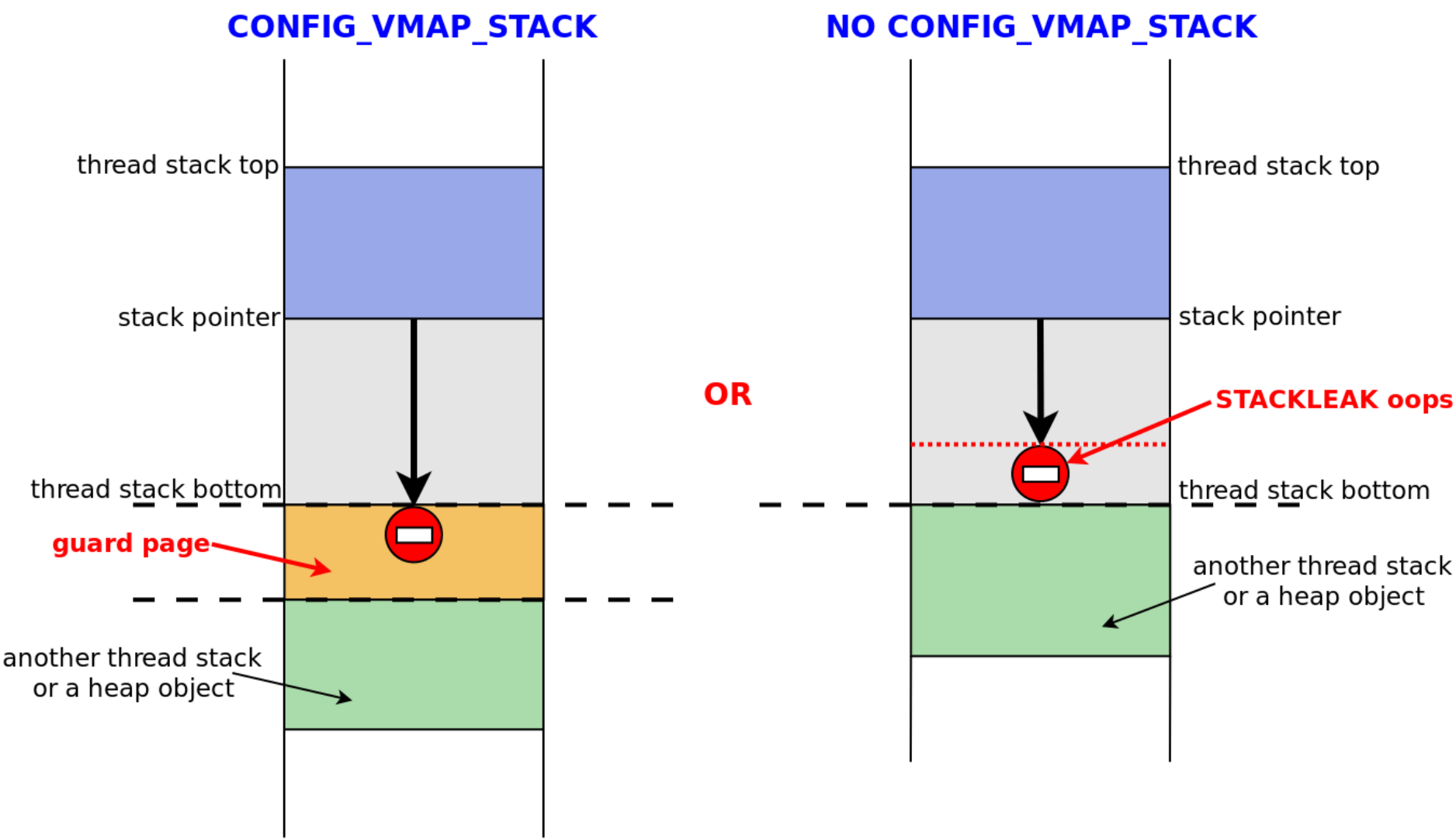
http://www.thegeekedgods.com/wp-content/uploads/2016/03/Empire-Strikes-Back.jpg

Kernel Stack Depth Overflow (2)

See "The Stack is Back" by Jon Oberheide (2012) and
"Exploiting Recursion in the Linux Kernel" by Jann Horn (2016)



CONFIG_VMAP_STACK or STACKLEAK Protection



STACKLEAK Demo #1

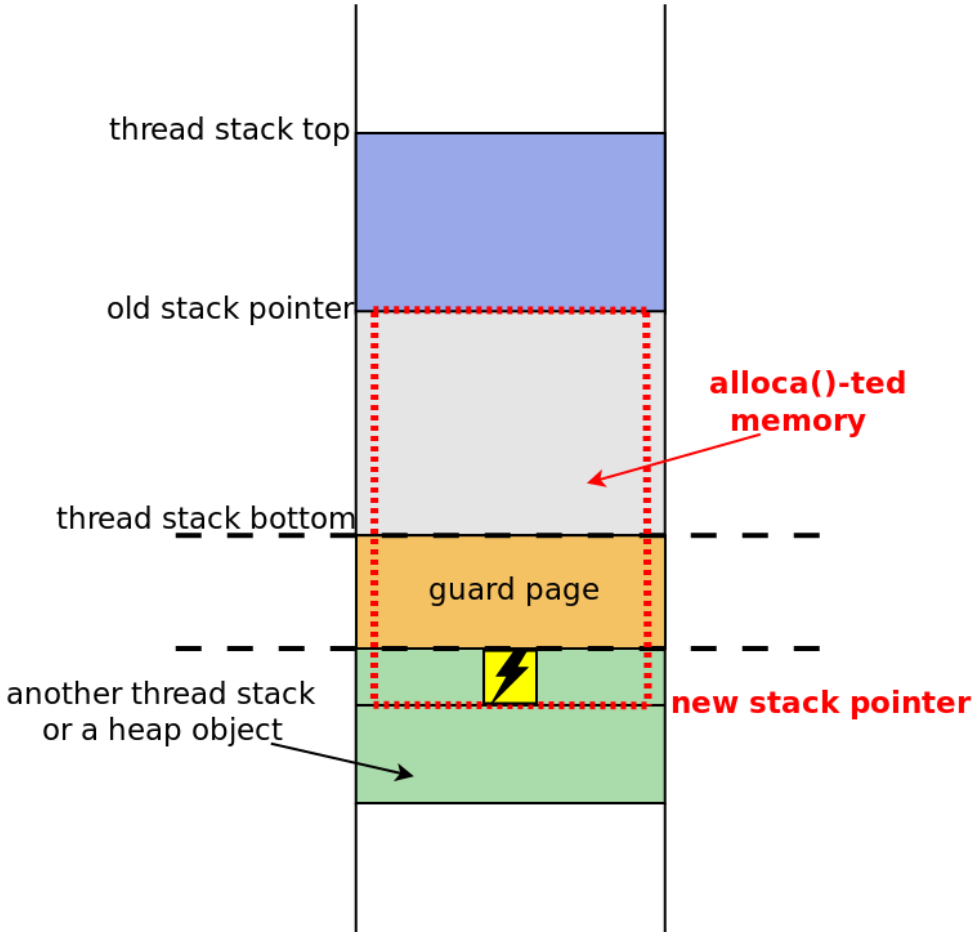
DEMO

Stack Clash Attack for the Kernel Stack

http://hacktext.com/seo201/lib/imgs/darth-vader-force-choke.jpg

Stack Clash Attack for the Kernel Stack

Idea by Gael Delalleau: "Large memory management vulnerabilities" (2005)
Revisited in "The Stack Clash" by Qualys Research Team (2017)



STACKLEAK Protection

- Read about STACKLEAK vs Stack Clash on grsecurity blog: https://grsecurity.net/an_ancient_kernel_hole_is_not_closed.php
- This code runs before each `alloca` call:

```
void __used check_alloca(unsigned long size)
{
    unsigned long sp = (unsigned long)&sp;
    struct stack_info stack_info = {0};
    unsigned long visit_mask = 0;
    unsigned long stack_left;

    BUG_ON(get_stack_info(&sp, current,
                          &stack_info, &visit_mask));
    stack_left = sp - (unsigned long)stack_info.begin;
    BUG_ON(stack_left < 256 || size >= stack_left - 256);
}
```

STACKLEAK Demo #2

DEMO

Cool, But What's the Price? (1)

Brief performance testing on x86_64
Hardware: Intel Core i7-4770, 16 GB RAM
Test 1, attractive: building the Linux kernel with Ubuntu config

```
time make -j9
```

```
Result on 4.11-rc8:

    real 32m14.893s
    user 237m30.886s
    sys 11m12.273s

Result on 4.11-rc8+stackleak:

    real 32m26.881s (+0.62%)
    user 238m38.926s (+0.48%)
    sys 11m36.426s (+3.59%)
```

Cool, But What's the Price? (2)

Brief performance testing on x86_64

Hardware: Intel Core i7-4770, 16 GB RAM

Test 2, UNattractive:

```
hackbench -s 4096 -l 2000 -g 15 -f 25 -P
```

```
Average on 4.11-rc8:  8.71s

Average on 4.11-rc8+stackleak:  9.08s (+4.29%)
```

Cool, But What's the Price? (3)

Conclusions

1. The performance penalty **varies** for different workloads
2. Test STACKLEAK on your expected workload before deploying in production

STACKLEAK Inner Workings

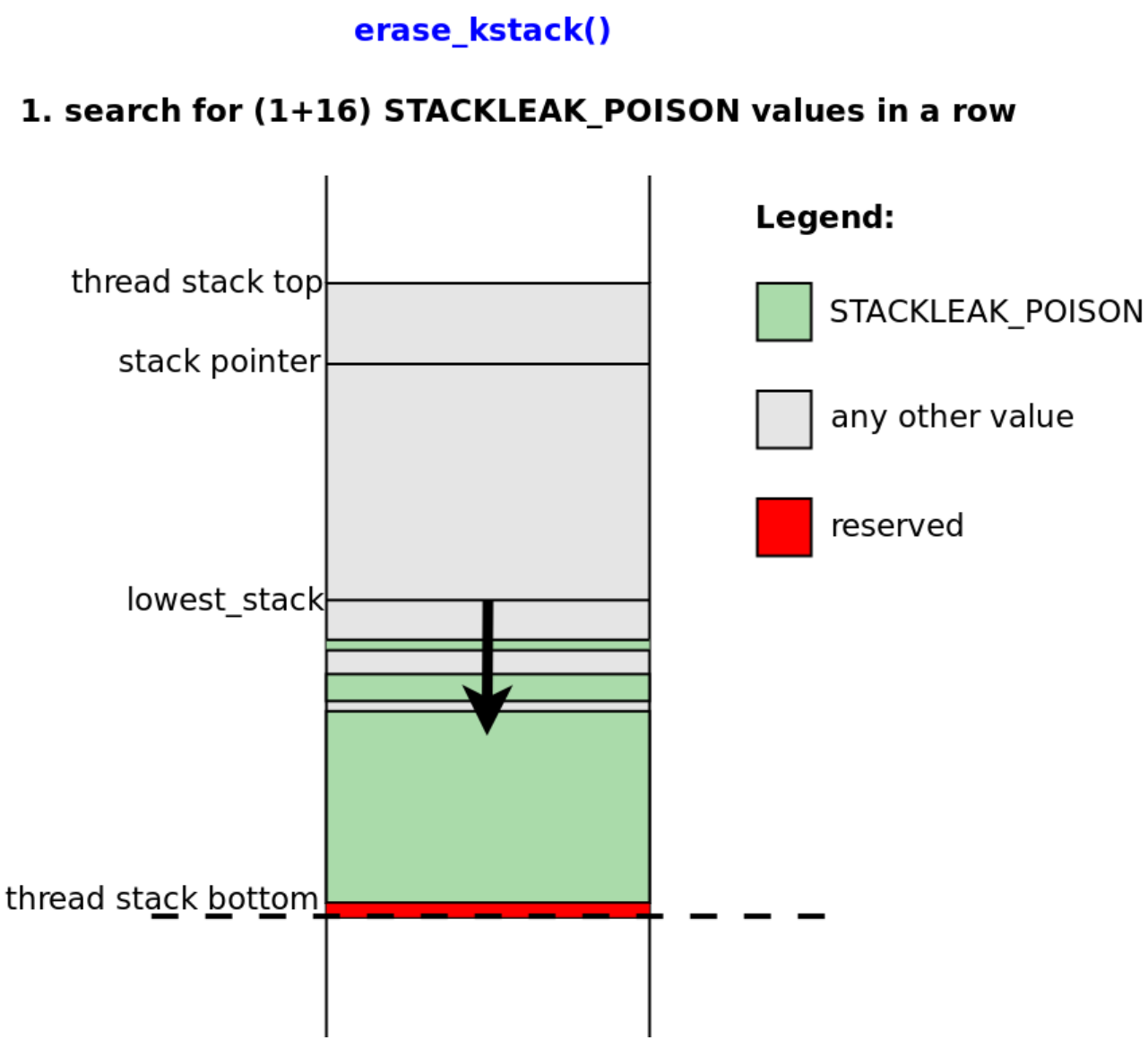
The STACKLEAK feature consists of:

- The asm code erasing the kernel stack
- The GCC plugin for compile-time instrumentation

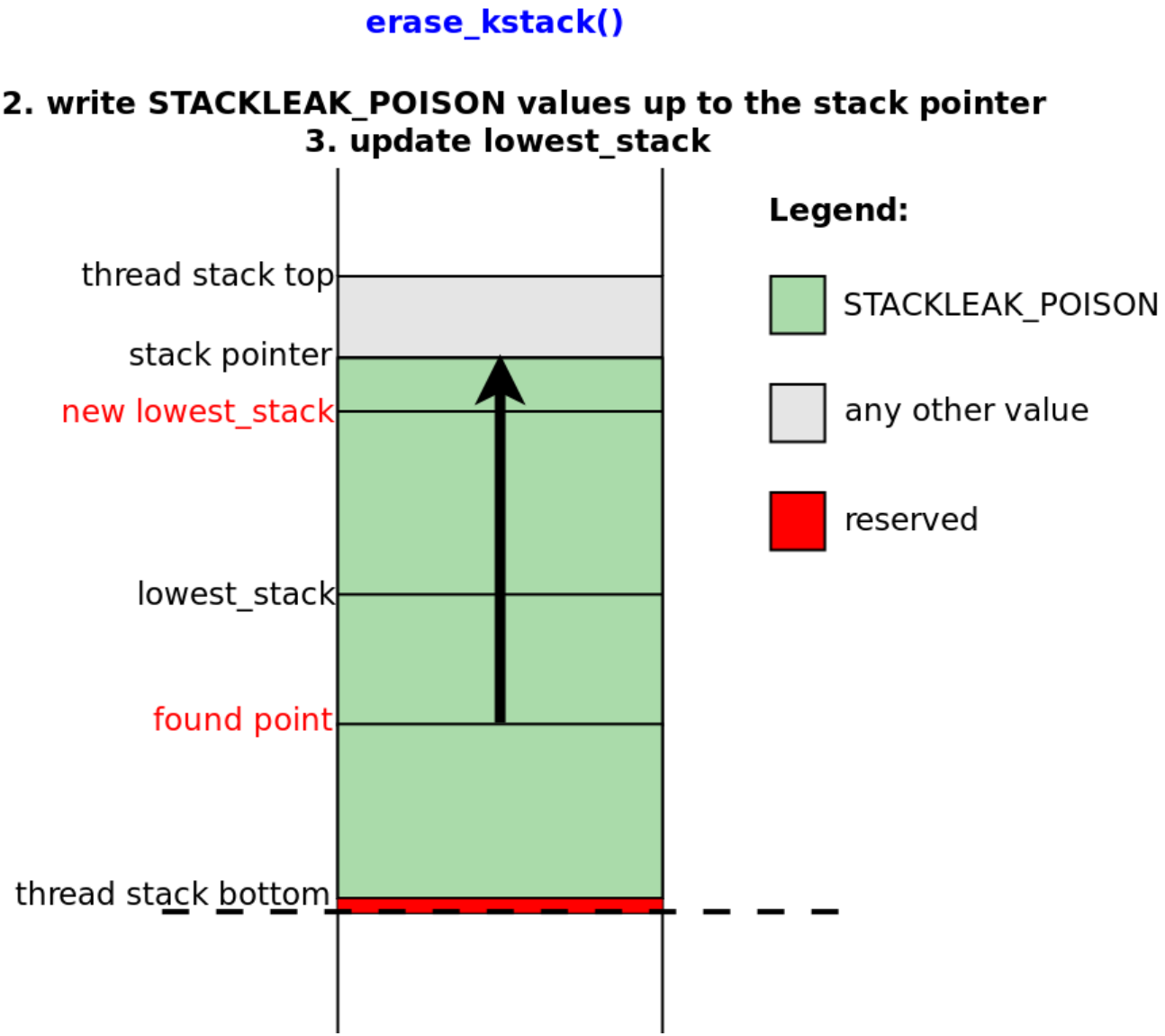
Erasing the Kernel Stack (1)

- The architecture-specific `erase_kstack()` function
- Works before returning from a syscall to userspace
- Writes `STACKLEAK_POISON` to the used part of the thread stack
- Uses `lowest_stack` updated by `track_stack()` as a starting point

Erasing the Kernel Stack (2)



Erasing the Kernel Stack (3)



Kernel Compile-Time Instrumentation

- Is done by STACKLEAK GCC plugin
- Inserts `track_stack()` call for the functions with a **big stack frame**
- Inserts the `check_alloca()` call before `alloca` and `track_stack()` call after it

GCC Plugins

- Are compiler loadable modules
- Are project-specific
- Register new passes via the GCC Pass Manager
- Provide the callbacks for these passes
- See wonderful slides by Diego Novillo:
  https://www.airs.com/dnovillo/200711-GCC-Internals/

STACKLEAK GCC Plugin

- Inserts function calls (complex operation)
- **But** needs to know the stack frame size (available too late)
- Nice solution by PaX Team!
- Registers 2 passes working with the IR of the code:

  1. stackleak_tree_instrument inserts function calls to GIMPLE
  2. stackleak_final removes them from RTL depending on the stack frame size

STACKLEAK Instrumentation Statistics

- For `x86_64_defconfig`
- The readelf utility shows 45602 functions in vmlinux
- STACKLEAK instrumented 2.853% of them
- The plugin inserted:
  - 36 `check_alloca()` calls,
  - 1265 `track_stack()` calls:
    - 42274 calls are inserted during GIMPLE pass
    - 41009 calls are deleted during RTL pass

My Final Propaganda

- WE are the Linux Kernel Community
- WE are responsible for servers, laptops, phones, PLCs, laser cutters and other crazy things running GNU/Linux
- Let's put some effort into **Linux Kernel Security**!

Thanks! Questions?

alex.popov@linux.com
@a13xp0p0v

http://blog.ptsecurity.com/
@ptsecurity