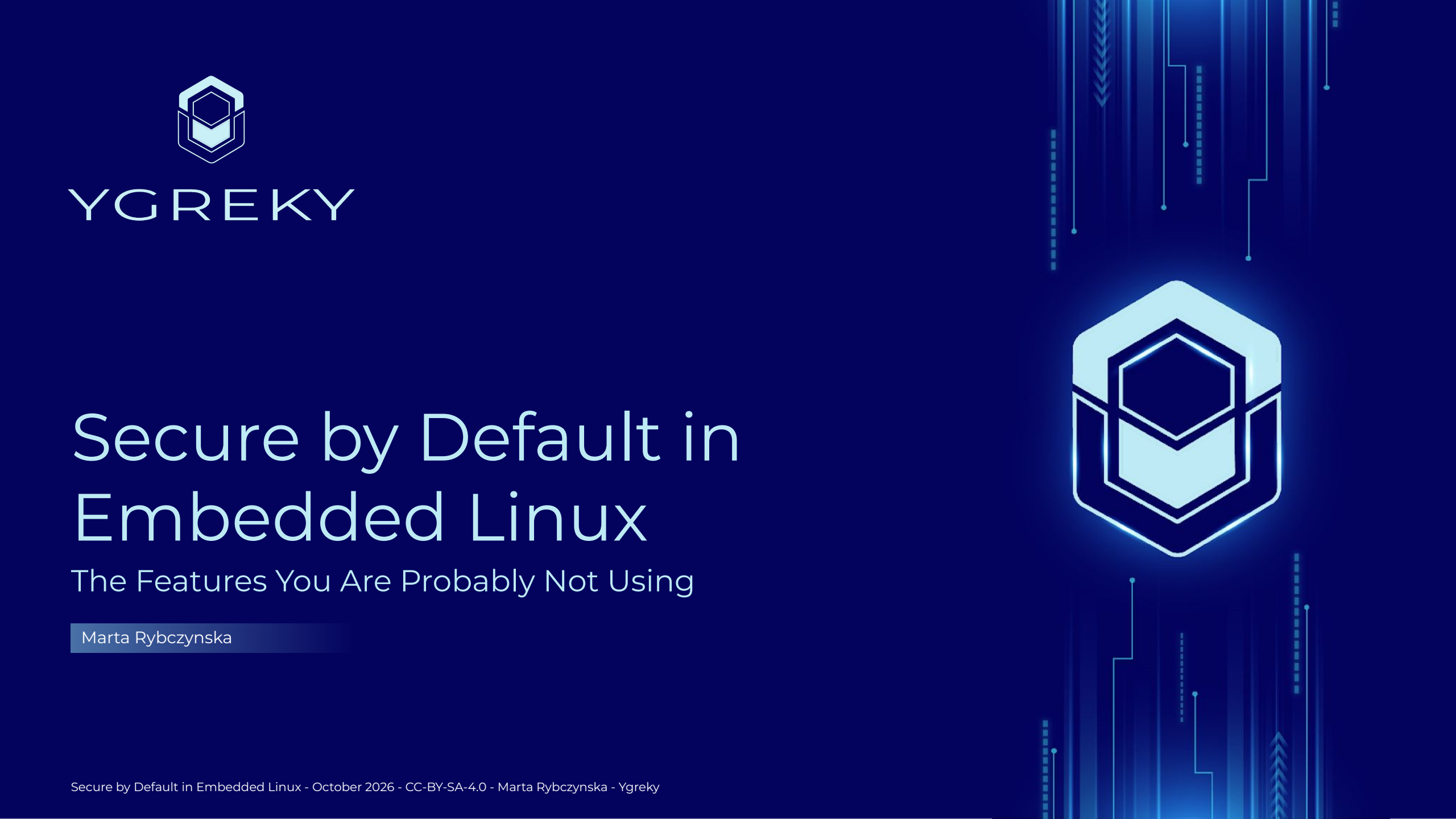



# Secure by Default in Embedded Linux

## The Features You Are Probably Not Using

Marta Rybczynska

# Secure by default - a requirement
## But this is not a Cyber Resilience Act talk

YGREKY



Official Journal
of the European Union

EN

L series

2024/2847

20.11.2024

**REGULATION (EU) 2024/2847 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL**

**of 23 October 2024**

**on horizontal cybersecurity requirements for products with digital elements and amending Regulations (EU) No 168/2013 and (EU) No 2019/1020 and Directive (EU) 2020/1828 (Cyber Resilience Act)**

**(Text with EEA relevance)**

THE EUROPEAN PARLIAMENT AND THE COUNCIL OF THE EUROPEAN UNION,

Having regard to the Treaty on the Functioning of the European Union, and in particular Article 114 thereof,

Having regard to the proposal from the European Commission,

After transmission of the draft legislative act to the national parliaments,

Having regard to the opinion of the European Economic and Social Committee ([1]),

After consulting the Committee of the Regions,

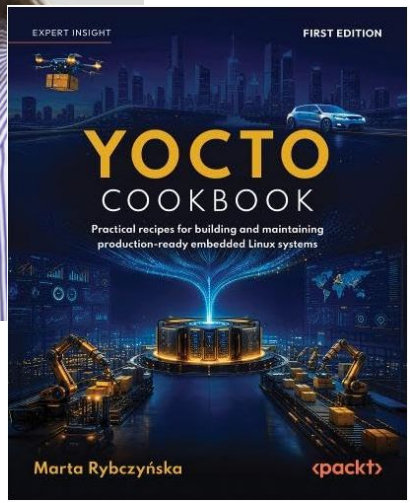
Acting in accordance with the ordinary legislative procedure ([2]),

# Who is Marta Rybczynska?

YGREKY

- PhD in Telecommunications
  - Network security/anonymity systems
- Open source/embedded developer/architect
  - 20+ years in open source
  - Contributions to the Linux kernel, various RTOSes
  - Involved into Yocto Project security
- Founder/CEO of Ygreky
  - Consulting (processes, architecture, reviews)
  - Teaching (“Embedded Security”, cybersecurity on-site courses and webinars)
  - Contributing to CRA-related standardisation
  - Author of the “CRA Readiness Program”





# What we have in the Linux kernel

YGREKY

docs.kernel.org/security/index.html

## Security Documentation

- Credentials in Linux
- Confidential Computing in Linux for x86 virtualization
- IMA Template Management Mechanism
- IMA Measurements Export and Delete
- Kernel Keys
- Linux Security Modules: General Security Hooks for Linux
- Linux Security Module Development
- Linux Secure Attention Key (SAK) handling
- SCTP
- Kernel Self-Protection
- SipHash - a short input PRF
- HalfSipHash - SipHash's insecure younger cousin
- Trusted Platform Module documentation
- Digital Signature Verification API
- Landlock LSM: kernel documentation
- Secrets documentation
- Integrity Policy Enforcement (IPE) - Kernel Documentation

Linux Kernel Self-Protection Project

### Mission Statement

Founded in 2015, this upstream Linux kernel project starts with the premise that kernel bugs have a very long lifetime, and that the kernel must be designed in ways to protect against these flaws. We must think of security beyond fixing bugs. As a community, we already find and fix individual bugs via static checkers (compiler flags, smatch, coccinelle, coverity, CodeQL) and dynamic checkers (kernel configs, syzkaller, KASan, trinity). Those efforts are important and on-going, but if we want to protect our billion Android phones, our cars, the International Space Station, and everything else running Linux, we must get proactive defensive technologies built into the upstream Linux kernel. We need the kernel to fail safely, instead of just running safely.

These kinds of protections have existed for years in out-of-tree projects and in piles of academic papers. For various social, cultural, and technical reasons, they have not made their way into the upstream kernel, and this project seeks to change that. Our focus is on kernel self-protection, rather than kernel-supported userspace protections. The goal is to eliminate classes of bugs and eliminate methods of exploitation.

### Security-related interfaces

- No New Privileges Flag
- Seccomp BPF (SECure COMPuting with filters)
- Landlock: unprivileged access control
- Linux Security Modules
- Introduction of non-executable mfd
- Speculation Control
- TEE (Trusted Execution Environment) Userspace API
- Executability check

### kernel.org Added as CVE Numbering Authority (CNA)

News February 13, 2024

kernel.org is now a CVE Numbering Authority (CNA) for any vulnerabilities in the Linux kernel as listed on kernel.org, excluding end-of-life (EOL) versions.



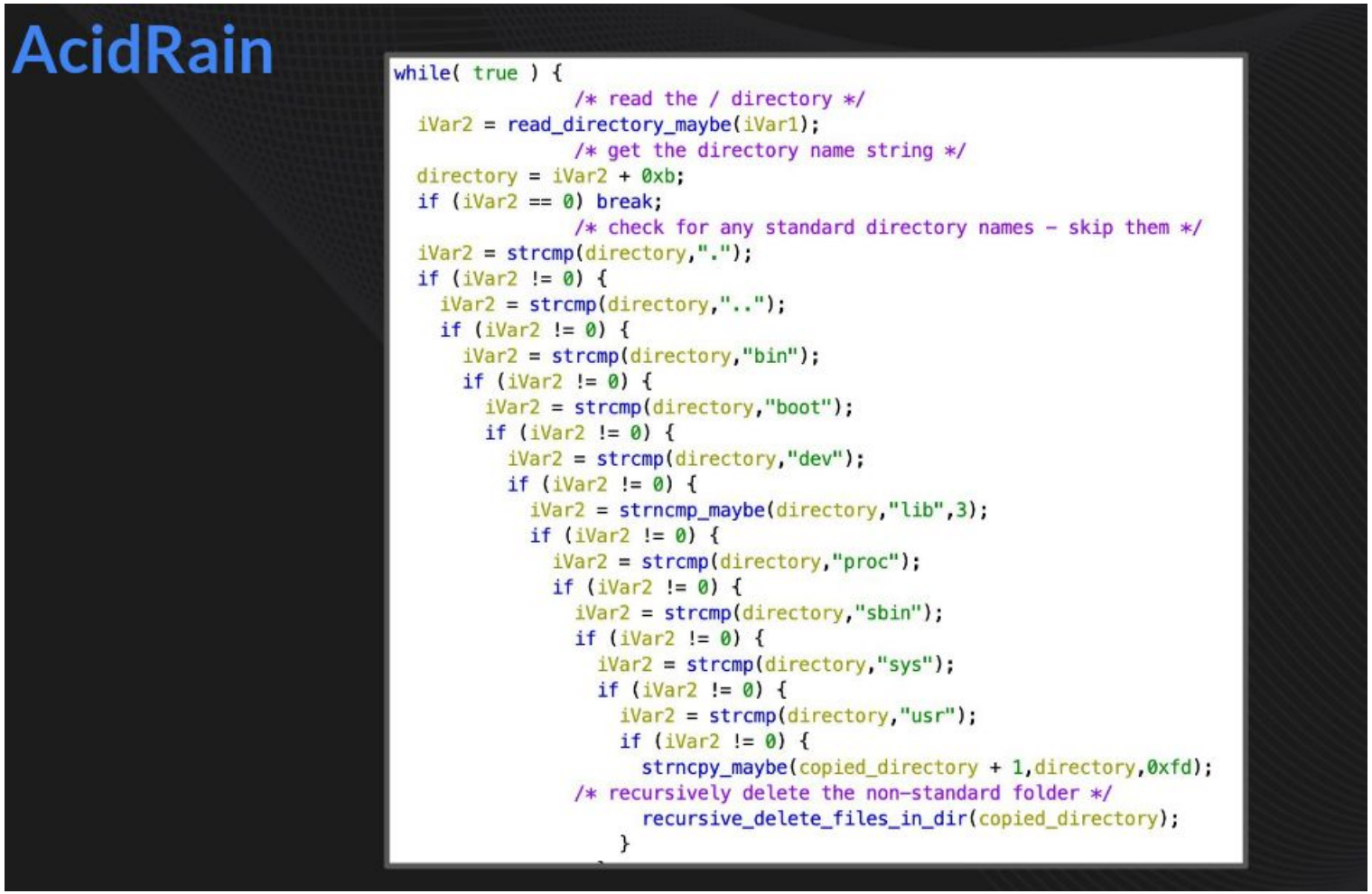
# Isn’t it “done”?



# At the same time, here's what we have in embedded...
## An example of embedded wiper malware

YGREKY

- The slide comes from "Real 'Cyber War': Espionage, DDoS, Leaks, and Wipers in the Russian Invasion of Ukraine" by Tom Hegel and Juan Andres Guerrero-Saade, Blackhat 2022 https://i.blackhat.com/USA-22/Wednesday/US-22-Hegel-Real-Cyber-War.pdf



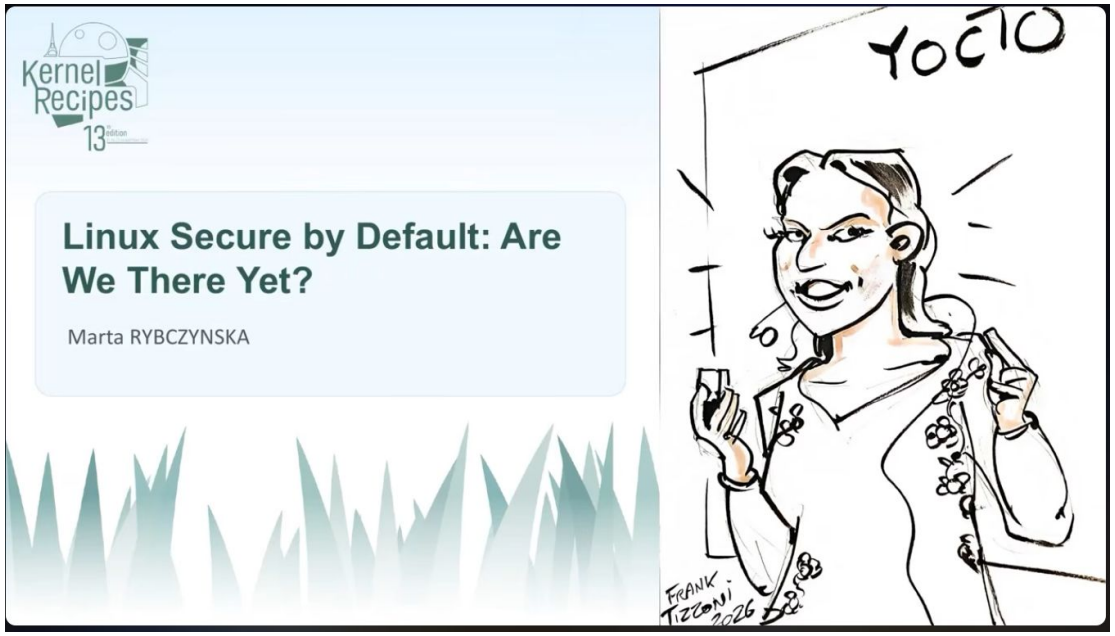
# Why?



# Conclusions from “Linux Secure by Default”
## At Kernel Recipes 2026

YGREKY

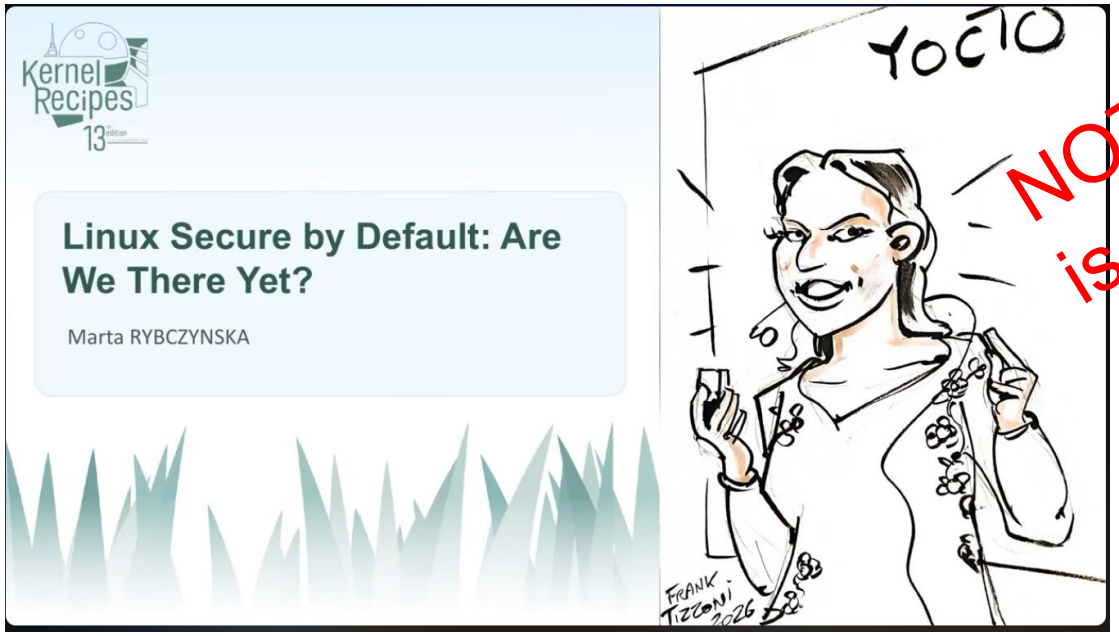
- Kernel security options are there
  - Security recommendations are also there (KSPP)
  - But security options NOT enabled by default (mostly)
  - Embedded distributions enable them selectively
- See the video:
  - https://youtu.be/MJXXaUAPHy0?si=oYCn4KhGmeIEF-Rl



# Conclusions from “Linux Secure by Default”
## At Kernel Recipes 2026

YGREKY

- Kernel security options are there
  - Security recommendations are also there (KSPP)
  - But security options NOT enabled by default (mostly)
  - Embedded distributions enable them selectively
- See the video:
  - https://youtu.be/MJXXaUAPHy0?si=oYCn4KhGmeIEF-Rl




NOTE: The kernel is not an exception.

Most open source projects are NOT secure by default

# And now: why?

## How many of you know how to configure:

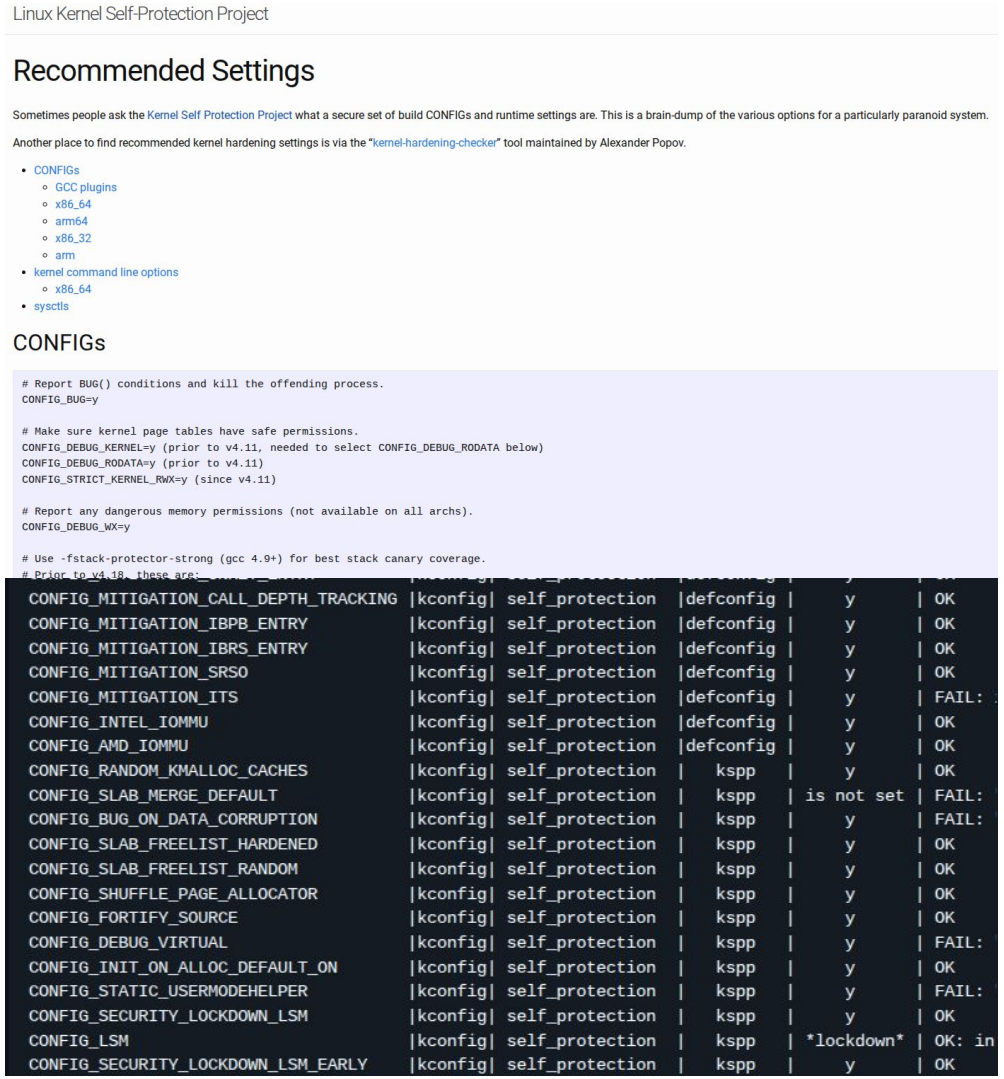



- Overlayfs
- Seccomp
- Fs-verity (unfortunately had to drop it - no time)
- Fscrypt
- Any LSM (Linux Security Module)

# Kernel hardening
## Hundreds of useful options out there

YGREKY

- Upstream resources:
  - kernel/config/hardening.config in the source tree
    - A limited set of low performance impact hardening options
  - Kernel Self Protect project - Recommended Settings
    - Includes kernel compilation options and sysctl settings
    - https://kspp.github.io/Recommended_Settings
  - Kernel-hardening-checker
    - Verifies a given kernel config with various recommendation sets (including KSPP)
    - Tool location: https://github.com/a13xp0p0v/kernel-hardening-checker



Linux Kernel Self-Protection Project

Recommended Settings

Sometimes people ask the Kernel Self Protection Project what a secure set of build CONFIGs and runtime settings are. This is a brain-dump of the various options for a particularly paranoid system.

Another place to find recommended kernel hardening settings is via the "kernel-hardening-checker" tool maintained by Alexander Popov.

- CONFIGs
  - GCC plugins
  - x86_64
  - arm64
  - x86_32
  - arm
- kernel command line options
  - x86_64
- sysctls

CONFIGs

```
# Report BUG() conditions and kill the offending process.
CONFIG_BUG=y

# Make sure kernel page tables have safe permissions.
CONFIG_DEBUG_KERNEL=y (prior to v4.11, needed to select CONFIG_DEBUG_RODATA below)
CONFIG_DEBUG_RODATA=y (prior to v4.11)
CONFIG_STRICT_KERNEL_RWX=y (since v4.11)

# Report any dangerous memory permissions (not available on all archs).
CONFIG_DEBUG_WX=y

# Use -fstack-protector-strong (gcc 4.9+) for best stack canary coverage.
# Prior to v4.18, these are:
```

```
CONFIG_MITIGATION_CALL_DEPTH_TRACKING |kconfig| self_protection  |defconfig |     y      | OK
CONFIG_MITIGATION_IBPB_ENTRY          |kconfig| self_protection  |defconfig |     y      | OK
CONFIG_MITIGATION_IBRS_ENTRY          |kconfig| self_protection  |defconfig |     y      | OK
CONFIG_MITIGATION_SRSO                |kconfig| self_protection  |defconfig |     y      | OK
CONFIG_MITIGATION_ITS                 |kconfig| self_protection  |defconfig |     y      | FAIL:
CONFIG_INTEL_IOMMU                    |kconfig| self_protection  |defconfig |     y      | OK
CONFIG_AMD_IOMMU                      |kconfig| self_protection  |defconfig |     y      | OK
CONFIG_RANDOM_KMALLOC_CACHES          |kconfig| self_protection  |   kspp   |     y      | OK
CONFIG_SLAB_MERGE_DEFAULT             |kconfig| self_protection  |   kspp   | is not set | FAIL:
CONFIG_BUG_ON_DATA_CORRUPTION         |kconfig| self_protection  |   kspp   |     y      | FAIL:
CONFIG_SLAB_FREELIST_HARDENED         |kconfig| self_protection  |   kspp   |     y      | OK
CONFIG_SLAB_FREELIST_RANDOM           |kconfig| self_protection  |   kspp   |     y      | OK
CONFIG_SHUFFLE_PAGE_ALLOCATOR         |kconfig| self_protection  |   kspp   |     y      | OK
CONFIG_FORTIFY_SOURCE                 |kconfig| self_protection  |   kspp   |     y      | OK
CONFIG_DEBUG_VIRTUAL                  |kconfig| self_protection  |   kspp   |     y      | FAIL:
CONFIG_INIT_ON_ALLOC_DEFAULT_ON       |kconfig| self_protection  |   kspp   |     y      | OK
CONFIG_STATIC_USERMODEHELPER          |kconfig| self_protection  |   kspp   |     y      | FAIL:
CONFIG_SECURITY_LOCKDOWN_LSM          |kconfig| self_protection  |   kspp   |     y      | OK
CONFIG_LSM                            |kconfig| self_protection  |   kspp   | *lockdown* | OK: in
CONFIG_SECURITY_LOCKDOWN_LSM_EARLY    |kconfig| self_protection  |   kspp   |     y      | OK
```

# Kernel hardening
## Hundreds of useful options out there

- Yocto Project solutions:
  - https://git.yoctoproject.org/yocto-kernel-cache/tree/features/security
  - Specific options for ARM64 and x86
- YP-style additional kernel fragment examples:
  - https://gitlab.com/ygreky/public/meta-mysecuredistro-example/-/tree/main/recipes-kernel/linux?ref_type=heads

yocto PROJECT · index : yocto-kernel-cache

Patches and configuration for the linux-yocto kernel tree

about summary refs log tree commit diff stats

path: root/features/security/security.cfg

blob: ea9be8853a560ee4e3dad448c13fbe4b26233b81 (plain)

```
1  # SPDX-License-Identifier: MIT
2  # Protect against ioctl buffer overflows
3  CONFIG_HARDENED_USERCOPY=y
4
5  # Check for memory copies that might overflow a structure in str*() and mem*()
6  # functions both at build-time and run-time
7  CONFIG_FORTIFY_SOURCE=y
8
9  # Harden the slab free list with randomization
10 CONFIG_SLAB_FREELIST_RANDOM=y
11 CONFIG_SLAB_FREELIST_HARDENED=y
12
13 # Stack Protector is for buffer overflow detection and hardening
14 CONFIG_STACKPROTECTOR=y
15 CONFIG_STACKPROTECTOR_STRONG=y
16
17 # Disable to ensure random heap placement to make exploits harder
18 CONFIG_COMPAT_BRK=n
19
20 # Disable; exposes kernel text image layout
21 CONFIG_PROC_KCORE=n
22
23 # Increases the low-level kernel attack surface. Disable it instead.
24 # Removes the modify_ldt system call.
25 CONFIG_EXPERT=y
26 CONFIG_MODIFY_LDT_SYSCALL=n
```

# Overlayfs
## Use case: full image update

YGREKY

- Function
  - Mount one file system on another
  - Useful if base file system is read only, can have writable directories or files
- Typical use
  - Writable /etc on read-only file systems
  - Writable configuration files
- Requires appropriate partitioning, example:
  - / - read-only
  - /data/ - writable user data
  - /tmp/ - temporary writable
  - /var/ - separate writable
  - Including A/B rootfs slots

# Overlayfs

## Example usage in YP-based images - /etc/ only - for testing (no partitioning)




```
EXTRA_IMAGE_FEATURES:append = " read-only-rootfs overlayfs-etc"
# Doesn’t work on read-only file systems
IMAGE_FEATURES:remove = "package-management"

# Ensure OverlayFS is built into the kernel.
# It must be built-in because overlayfs-etc runs before normal module
loading.
KERNEL_EXTRA_FEATURES:append = " features/overlayfs/overlayfs.scc"

# Volatile writable layer for /etc
OVERLAYFS_ETC_MOUNT_POINT = "/data"
OVERLAYFS_ETC_DEVICE = "tmpfs"
OVERLAYFS_ETC_FSTYPE = "tmpfs"
OVERLAYFS_ETC_MOUNT_OPTIONS = "mode=0755"
```

# Seccomp

## Use case: I have a legacy app I can't change




- Allows blocking selected system calls for a given application
- Advantages
  - Can block malware
  - Also unknown
- Difficulties
  - Need to profile each application separately

# Seccomp
## What do we need: an example simple loader (here with libseccomp)

YGREKY

```
#include <seccomp.h>
[...]

static int add_policy_file(scmp_filter_ctx ctx, const
char *filename)
{
       /* read the config file line by line, then for
any syscall name p ...*/

        nr = seccomp_syscall_resolve_name(p);

        if (nr == __NR_SCMP_ERROR) {
            fprintf(stderr, "Unknown syscall in policy:
%s\n", p);
            fclose(f);
            return -1;
        }

        if (seccomp_rule_add(ctx, SCMP_ACT_ALLOW, nr, 0)
< 0) {
            fprintf(stderr, "Cannot add syscall: %s\n",
p);
            fclose(f);
            return -1;
        }
    }

   /* .... */
}
```

```
int main(int argc, char **argv)
{
    scmp_filter_ctx ctx;

    /*...*/

    ctx = seccomp_init(SCMP_ACT_ERRNO(EPERM));
    if (!ctx) {
        fprintf(stderr, "seccomp_init failed\n");
        return EXIT_FAILURE;
    }

    if (add_policy_file(ctx, argv[1]) < 0) {
        seccomp_release(ctx);
        return EXIT_FAILURE;
    }

    if (seccomp_load(ctx) < 0) {
        perror("seccomp_load");
        seccomp_release(ctx);
        return EXIT_FAILURE;
    }

    execvp(argv[2], &argv[2]);

    perror("execvp");
    return EXIT_FAILURE;
}
```

# Seccomp
## What do we need: syscall profile

YGREKY

Legacy-app.syscalls:

```
access
arch_prctl
brk
close
execve
exit_group
fstat
getrandom
ioctl
mmap
mprotect
munmap
newfstatat
openat
pread64
prlimit64
read
rseq
set_robust_list
set_tid_address
write
```

- Profiling your application:
  - strace
- Then run the application using the runner
  - Syscall not on the list will cause EPERM

# Seccomp
## What do we need: application syscall profile

YGREKY

Legacy-app.syscalls:

```
access
arch_prctl
brk
close
execve
exit_group
fstat
getrandom
ioctl
mmap
mprotect
munmap
newfstatat
openat
pread64
prlimit64
read
rseq
set_robust_list
set_tid_address
write
```

- Profiling your application:
  - strace
- Then run the application using the runner
  - Syscall not on the list will cause EPERM
- Risks/harder problems
  - Need to profile all use cases
  - Some syscall are more risky than others (ioctl !)

# fscrypt
## Per-user encryption




- Function
  - Use filesystem encryption, at a directory level
- Advantaged
  - Can support different keys by different users
- Difficulties
  - Key management - need secure storage

# fscrypt
## Configuring the filesystem encryption

YGREKY

- Create your filesystem with the encryption option
  - `mkfs.ext4 -F -O encrypt /tmp/fscrypt.img`

```
root@qemux86-64:~# tune2fs -l /tmp/fscrypt.img
tune2fs 1.47.4 (6-Mar-2025)
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          27a07542-6422-4eb7-b933-859bb93c16ec
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index orphan_file filetype needs_recovery extent 64bit flex_bg metadata_csum_seed encrypt sparse_super
Filesystem flags:         signed_directory_hash
```

- Create and add a key
  - `fscryptctl add_key /mnt/data < keyfile`
  - Note the key ID!

# fscrypt
## Setting up the key

YGREKY

- Set up the key for the specific directory

```
root@qemux86-64:~# fscryptctl set_policy "$KEYID" /mnt/data/private
root@qemux86-64:~# fscryptctl get_policy /mnt/data/private/
Encryption policy for /mnt/data/private/:
        Policy version: 2
        Master key identifier: 50a426734afb49f95cd932763a9b7bf5
        Contents encryption mode: AES-256-XTS
        Filenames encryption mode: AES-256-CTS
        Flags: PAD_32
```

# fscrypt
## Setting up the key

YGREKY

- Write data to the file system

```
root@qemux86-64:~# echo "secret" > /mnt/data/private/secret.txt
[  518.756579] fscrypt: AES-256-CBC-CTS using implementation "cts(cbc(ecb(aes-generic)))"
[  518.801105] fscrypt: AES-256-XTS using implementation "xts(ecb(aes-generic))"
root@qemux86-64:~# cat /mnt/data/private/secret.txt
secret
root@qemux86-64:~# echo "pubic" > /mnt/data/public.txt
root@qemux86-64:~# cat /mnt/data/public.txt
pubic
root@qemux86-64:~# sync
root@qemux86-64:~# umount /mnt/data/
```

# fscrypt
## Setting up the key

YGREKY

- When mounting the file system again...

```
root@qemux86-64:~# mount -o loop /tmp/fscrypt.img  /mnt/data/
[  702.825641] loop0: detected capacity change from 0 to 131072
[  702.843587] EXT4-fs (loop0): mounted filesystem 8d08204e-5cec-4868-a88f-8c31c377992a r/w with ordered data mode. Quota mode: disabled.
root@qemux86-64:~# cat /mnt/data/public.txt
pubic
root@qemux86-64:~# cat /mnt/data/private/MiHJPHDgGXMiUfcGhy87FdfdVlC-Vx2OVkaxZakjrr4PuQbwX5Bn8Q .
cat: can't open '/mnt/data/private/MiHJPHDgGXMiUfcGhy87FdfdVlC-Vx2OVkaxZakjrr4PuQbwX5Bn8Q': Required key not available
cat: read error: Is a directory
```

# fscrypt
## Setting up the key




- Add the key again:
  - Don’t use /tmp/ for keys :)

```
root@qemux86-64:~# fscryptctl add_key /mnt/data < /tmp/fscrypt.key
50a426734afb49f95cd932763a9b7bf5
root@qemux86-64:~# cat /mnt/data/
lost+found/  private/     public.txt
root@qemux86-64:~# cat /mnt/data/private/secret.txt
secret
```

# There’s more

## There are many other options you may use



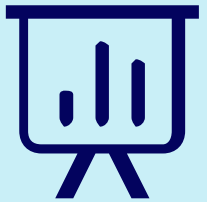

- Firewalls (yes, activate it even in a local network)
- Dm-crypt
- Running applications as dedicated users
- Any LSM (Linux Security Module)

# Conclusions
## Configuring security features is not that hard

YGREKY

- Good news
  - Configuring security features is not that hard
  - There’s a wide choice of options
- Bad news
  - Good examples are missing
  - Standard configurations of embedded distros frequently are quite far from “secure by default”
  - “Poky” is definitely not a production distro

YGREKY



# Discussion time

Contact:
marta.rybczynska@ygreky.com