Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

3: Compile Kernel

Hes·so FRIBOURG FREIBURG
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

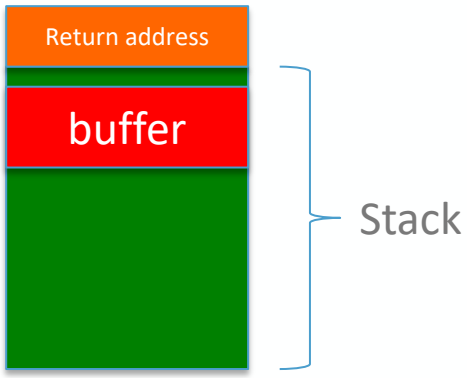
Securité IT | Jean-Roland Schuler, jean-roland.schuler@hefr.ch

Software attacks

Buffer overflow:

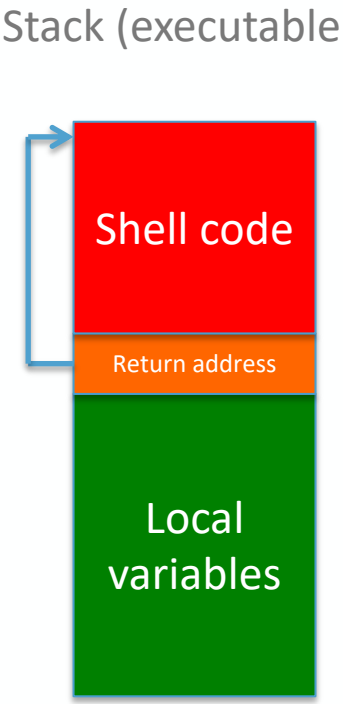
Example:

```
void main () {
char buffer [4];
    strcpy (&buffer[0], “123456”);
}
```



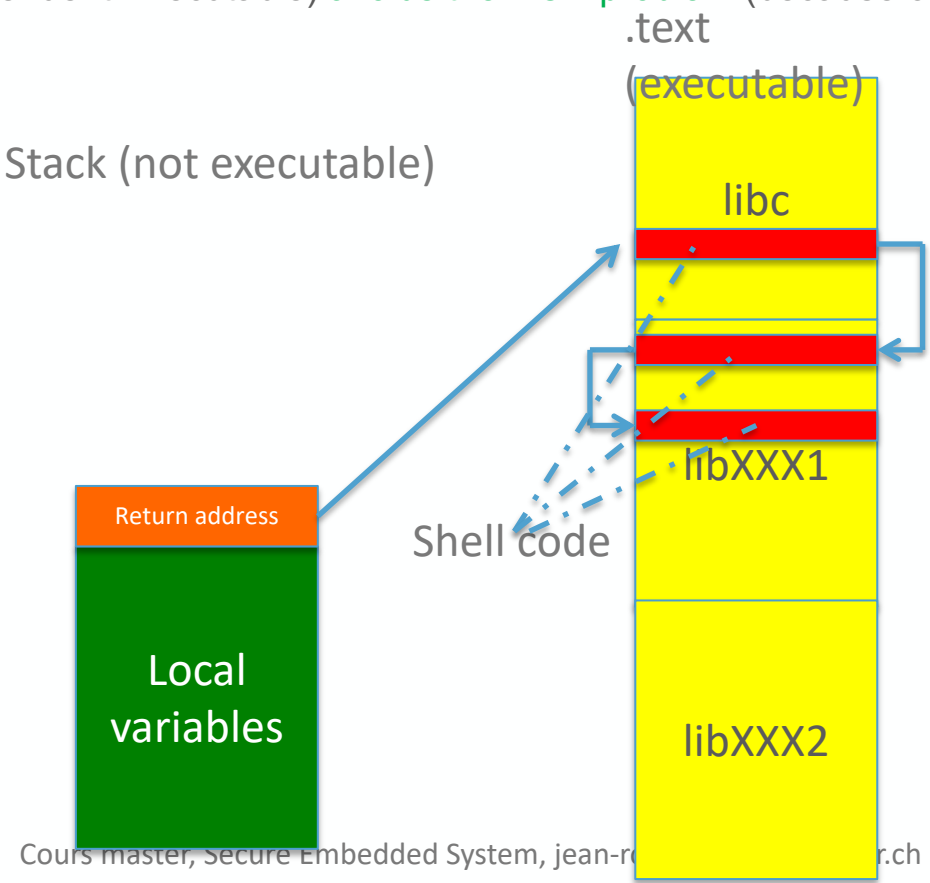
Software attacks

- A buffer overflow attack can insert and executes a shell code in an executable stack.



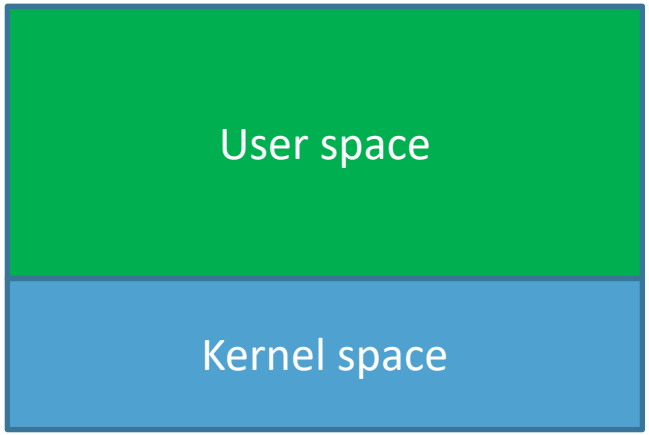
Software attacks

- Currently, the stack memory is no longer executable
- However, a technique called ret2libc could be used to bypass non executable memory. The main idea is to execute code in an executable memory like libc() or other libraries.
- Another technique called ROP, or Return-Oriented Programming allows also to bypass this protection. The main idea is to execute code in the program itself
- ASLR (Address Space Layout Randomization) randomize_va_space is used in order avoid the ret2lib (because stack and head addresses change)
- The PIE (Position Independent Executable) avoids the ROP problem (because code addresses change)



Linux spaces

- Generally users use the User space
- `dmesg, /dev/mem, /proc, /sys, modules and devices drivers` allow data exchange between User and Kernel spaces



Linux kernel

Linux kernel is in this directory:

```
~/workspace/nano/buildroot/output/build/linux-xx/
```

This directory has these main sub-directories:

arch	Hardware dependent code
block	Generic functions for the block devices
crypto	Cryptographic algo. used in the kernel
Documentation	Documentation about the kernel
drivers	All drivers known by the kernel
fs	All filesystem know by the kernel
include	kernel include files
init	Init code (function start_kernel)
ipc	Interprocess communication
kernel	Kernel code, scheduler, mutex, …
lib	different libraries used by the kernel

Linux kernel

```
mm              Memory management
net             Different protocols, IPv4, IPv6, bluetooth, ...
samples         Different examples, kobject, kfifo, ...
security        Encrypted keys, SELinux, ...
sound           Sound support for Linux kernel
virt            Kernel-based virtual machine
```

This directory has these main files

```
vmlinux         Linux kernel, ELF format, ARM aarch64
.config         Linux kernel configuration
.config.old     Old Linux kernel configuration
Kconfig         Configuration for the make linux-xconfig
Makefile        makefile
```

Boot Linux kernel

During the startup phase, Linux executes first this file:

```
arch/arm64/kernel/head.S                    // hardware dependent code
```

Next this function is called:

```
start_kernel( ) (init/main.c)  // hardware independent code
```

This function initializes different thinks:

```
Boot CPU
Memory management
SMP, Symmetric Multi-processing
Parse the boot parameters
Interruptions
Scheduler
Timers
Initial Ram Disk
Key
Security (SELinux, …)
```

make friendlyamr_nanopi_neo_plus2_defconfig

Important: cd ~/workspace/nano/buildroot

During buildroot installation the command

`make friendlyarm_nanopi_neo_plus2_defconfig` was executed

The `make friendlyarm_nanopi_neo_plus2_defconfig` command initializes the `.config` file and indicates which processor, card, hardware, … is used.

.config file:

```
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=80201
CONFIG_CLANG_VERSION=0
CONFIG_CC_HAS_ASM_GOTO=y
CONFIG_CC_HAS_WARN_MAYBE_UNINITIALIZED=y
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_EXTABLE_SORT=y
CONFIG_THREAD_INFO_IN_TASK=y

#
# General setup
#
CONFIG_INIT_ENV_ARG_LIMIT=32
# CONFIG_COMPILE_TEST is not set
CONFIG_LOCALVERSION=""
```

Compile linux

Compile

```
cd ~/workspace/nano/buildroot

make linux-menuconfig or     // to configure the linux kernel
make linux-xconfig           // to configure the linux kernel
make linux-rebuild           // to compile only the linux kernel

Other commands
make clean                     // Be careful: delete output
                               // directory and the .o files
                               // → compile all files
make help                      // Different commands
make nano_neno_plus2_defconfig // Only the first time
make V=1                       // print commands
```

make linux-xconfig

```
Important: cd ~/workspace/nano/buildroot
```

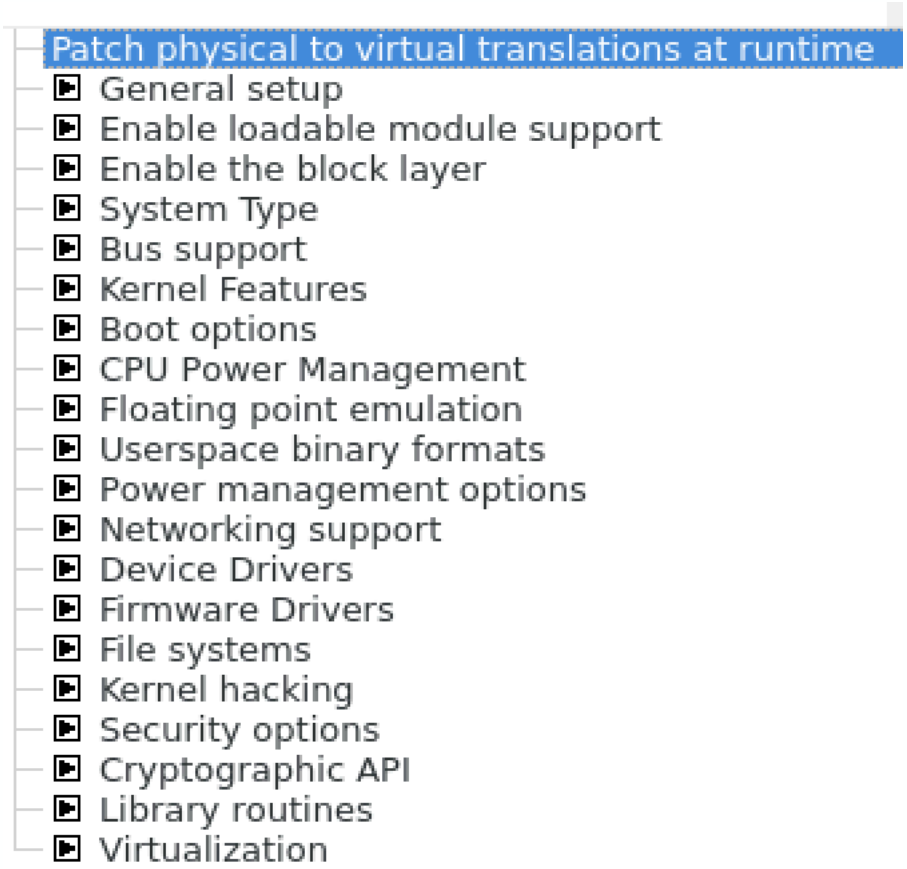
`make linux-xconfig or make linux-menuconfig` configures the kernel

```
make linux-xconfig      (remark: install qt and qt-devel)
```

Make linux-xconfig

Kernel configuration is organized in different categories

- General setup
- Enable loadable module ….
- Enable the block layer
- …



Disable Kernel .config support (CONFIG_IKCONFIG)

This option enables the complete Linux kernel ".config" file contents to be saved in the kernel. Kernel options can be read with this command: `cat /prog/config.gz`

Hardening Linux: Not include the configuration file inside the kernel

make linux-xconfig: General Setup → [ ] Kernel .config support

Enable -fstack-protector buffer overflow detection (CC_STACKPROTECTOR)

gcc `-fstack-protector-all` option adds extra code to check for buffer overflows, such as stack smashing attacks. This is done by adding a guard variable to functions.

Hardening Linux:

make linux-xconfig: General architecture-dependent options →

[*] Stack Protector buffer overflow detection

[*] Strong Stack Protector

Enabling this option provides some level of protection against stack based buffer overflows within the Linux kernel memory (not the user processes). If detected, the kernel will die with a kernel panic.

Enable -fstack-protector buffer overflow detection (CC_STACKPROTECTOR)

Example: file.c has an buffer overflow error, the –fstack-protector-all detects that

```
unsigned char localBufferOverflow (void) {
unsigned char val   [16];
int           i;
    for (i=0; i< 18; i++)
        val[i] = 0;
    return val[0];
}

# gcc -Wall -fstack-protector-all -o file file.C
#./file
*** stack smashing detected ***: ./file terminated
======= Backtrace: =========
/lib/libc.so.6(__fortify_fail+0x45)[0x4de0eb85]
/lib/libc.so.6[0x4de0eb3a]
```

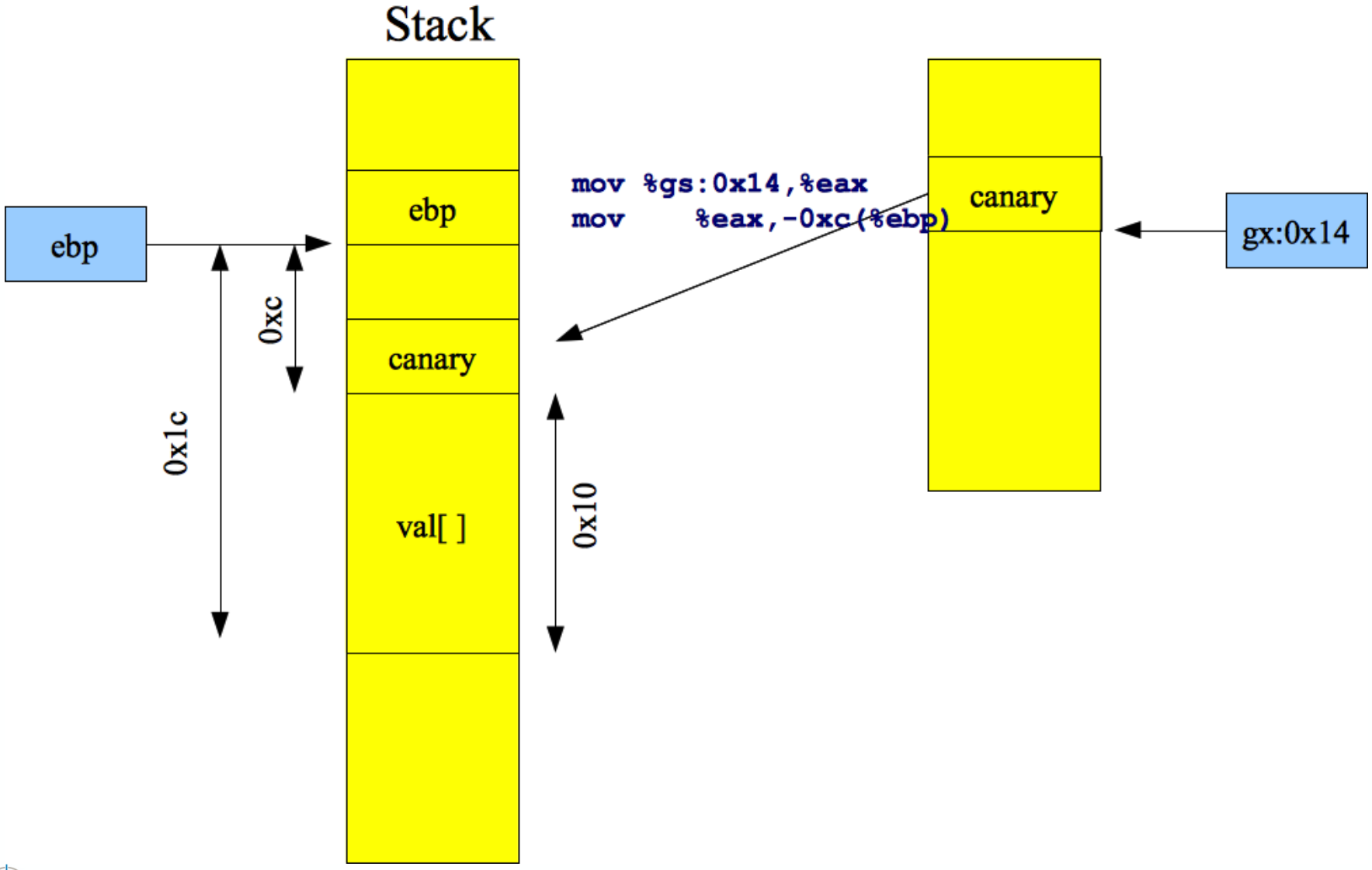
-Fstack-protection-all gcc option

Principe: localBufferOverflow assembler code (Intel code)

```
08048453 <_ZL19localBufferOverflowv>:
 8048453:       55                      push   %ebp
 8048454:       89 e5                   mov    %esp,%ebp
 8048456:       83 ec 28                sub    $0x28,%esp
 8048459:       65 a1 14 00 00 00       mov    %gs:0x14,%eax
 804845f:       89 45 f4                mov    %eax,-0xc(%ebp)
 8048462:       31 c0                   xor    %eax,%eax
 ....
 804848b:       8b 55 f4                mov    -0xc(%ebp),%edx
 804848e:       65 33 15 14 00 00 00    xor    %gs:0x14,%edx
 8048495:       74 05                   je     804849c
 8048497:       e8 54 fe ff ff          call   80482f0 <__stack_chk_fail@plt>
 804849c:       c9                      leave
```

If the edx register is not zero, the function __stack_chk_fail@plt is called

-Fstack-protection-all gcc option



-Fstack-protection-all gcc option, ARM uP

```
08048453 <testCanary>:
        1049c:       e58de004        str     lr, [sp, #4]
        104a0:       e28db004        add     fp, sp, #4
        104a4:       e24dd010        sub     sp, sp, #16
        104a8:       e3003f08        movw    r3, #3848       ; 0xf08
        104ac:       e3403002        movt    r3, #2
        104b0:       e5933000        ldr     r3, [r3]
        104b4:       e50b3008        str     r3, [fp, #-8]
        104b8:       e3a03000
        …
        104f8:       e3003f08        movw    r3, #3848       ; 0xf08
        104fc:       e3403002        movt    r3, #2
        10500:       e51b2008        ldr     r2, [fp, #-8]
        10504:       e5933000        ldr     r3, [r3]
        10508:       e1520003        cmp     r2, r3
        1050c:       0a000000        beq     10514 <testCanary+0x7c>
        10510:       ebffff8e        bl      10350 <__stack_chk_fail@plt>
        10514:       e24bd004        sub     sp, fp, #4
        10518:       e59db000        ldr     fp, [sp]
        1051c:       e28dd004        add     sp, sp, #4
        10520:       e49df004        pop     {pc}            ; (ldr pc, [sp],
```

If r2 != r3 --> __stack_chk_fail function is called

Randomize_va_space (CONFIG_COMPAT_BRK)

More information: <kernel source>/Documentation/sysctl/kernel.txt

Make linux-xconfig: General Setup → [ ] Disable Heap randomization

This option can be used to select the type of process address space randomization (Called: Address space layout randomization (ASLR))

0 - Turn the process address space randomization off. This is the default for architectures that do not support this feature anyways, and kernels that are booted with the "norandmaps" parameter.

1 - Make the addresses of mmap base, stack and VDSO (virtual dynamically linked shared objects) page randomized. This, among other things, implies that shared libraries will be loaded to random addresses. This is the default if the CONFIG_COMPAT_BRK option is enabled.

2 - Additionally enable heap randomization. This is the default if CONFIG_COMPAT_BRK is disabled.

```
On the nanoPi:
cat /proc/sys/kernel/randomize_va_space            // show
echo value > /proc/sys/kernel/randomize_va_space   // modify
sysctl -w kernel.randomize_va_space=value          // modify value=0,1,2
```

Randomize_va_space (CONFIG_COMPAT_BRK)

Different commands on nanoPi:

Show the randomize_va_space value:

```
cat /proc/sys/kernel/randomize_va_space
```

Modify the randomize_va_space value:

```
echo 2 > /proc/sys/kernel/randomize_va_space
sysctl -w kernel.randomize_va_space=0 or 1 or 2
```

Hardening Linux:

```
echo 1 > /proc/sys/kernel/randomize_va_space
```

Better:

```
echo 2 > /proc/sys/kernel/randomize_va_space
sysctl -w kernel.randomize_va_space=2
```

Randomize address of kernel Image
(CONFIG_RANDOMIZE_BASE)

Randomizes virtual address at which the kernel image is loaded, as a security feature that deters exploit attempts relying on knowledge of the location of kernel internals.

Hardening Linux:

make linux-xconfig:

Kernel feature → [*] Randomize the address of the kernel image

Randomize SLAB Allocator

(CONFIG_SLAB_FREELIST_RANDOM, CONFIG_SLAB_FREELIST_HARDENED)

Make linux-xconfig: General Setup

[Choose SLAB allocator (SLAB) → [*] SLAB
[*] Allow slab caches to be merged
[*] SLAB freelist randomization

Slab allocation is a memory management mechanism intended for the efficient memory allocation of kernel objects.o, it randomizes the freelist order used on creating new pages. This security feature reduces the predictability of the kernel slab allocator against heap overflows

Write protect kernel text and module
(CONFIG_RODATA_FULL_DEFAULT_ENABLED)

Kernel code is in the text section and this section must be read only.

This prevents code or read-only data from being modified (inadvertently or intentionally) via another mapping of the same memory page.

Hardening Linux:

make linux-xconfig:

Kernel Feature → [*] Apply r/o permissions of VM areas also to their linear aliases

Optimize for performance or size

(CONFIG_CC_OPTIMIZE_FOR_PERFORMANCE, CC_OPTIMIZE_FOR_SIZE)

It is possible to optimize the kernel size (gcc –Os) or the performance (gcc –O2)

But for the current embedded systems, it is not necessary to optimize the kernel size

Make linux-xconfig: General Setup → Compiler optimization level

→ [*] Optimize for performance

Enable random number generator

dev.gentoo.org/~swift/docs/security_benchmarks/kernel.html

Enable the random number generator to provide a secure source of random numbers (which is important for cryptographic functions).

This can be accomplished by using the CONFIG_ARCH_RANDOM setting and next to select the random generator for the platform

Make linux-xconfig:

Device drivers → Character Devices → [*] Hardware Random Number Generator Core support
[*] Timer IOMEM HW Random Number Generator support
[*] Broadcom BCM2835/BCM63xx Random Number Generator support

Enable random number generator

- If the hardware used has no random generator, it is possible to use the HAVEGE [HArdware Volatile Entropy Gathering and Expansion] service (https://www.irisa.fr/caps/projects/hipsor/).
- It is an unpredictable random number generator is a practical approximation of a truly random number generator.

```
Important: cd ~/workspace/nano/buildroot
make menuconfig
    → Target Packages → Miscellaneous → haveged
```

…

On nanopi: The entropy can be read:

```
cat /proc/sys/kernel/random/entropy-avail
sysctl kernel.random.entropy_avail
```

Filter access to /dev/mem (STRICT_DEVMEM)

Do not allow all processes full access to all of the systems' memory through /dev/mem (which includes kernel memory and process memory).

By enabling CONFIG_STRICT_DEVMEM. Root user can only access memory regions expected for all legitimate common usage

Hardening Linux:

make linux-xconfig: Kernel Hacking → [*] Filter access to /dev/mem

→ [*] Filter I/O access to /dev/mem

Strip assembler-generated symbols during link
(STRIP_ASM_SYMS)

Strip internal assembler-generated symbols during a link, it is harder to reverse the code

Hardening Linux:

make linux-xconfig:

Kernel Hacking → Compile time check and compiler options

→ [ ] Compile the kernel with debug info

→ [*] Strip assembler-generated symbols during link

Restrict unprivileged access to the kernel syslog (SECURITY_DMESG_RESTRICT)

Only root can access to the kernel system logs (through **dmesg**).

Hardening Linux:

make linux-xconfig: Security options → [*] Restrict unprivileged access to the kernel syslog

Kernel Memory Initialization (CONFIG_INIT_STACK_NONE, CONFIG_INIT_ON_ALLOC_DEFAULT_ON, CONFIG_INIT_ON_FREE_DEFAULT_ON)

This options enable initialization of stack variables and heap memory.

Hardening Linux:

make linux-xconfig: Security options → Kernel Hardening Options → Memory Initialization
→ Initialize Kernel Stack … → ( ) no automatic initialization (weakest) **(if possible)**
[*] Enable heap memory zeroing on allocation by default
[*] Enable heap memory zeroing on free by default

Check memory copies between kernel and unserspace (CONFIG_HARDENED_USERCOPY, CONFIG_HARDENED_USERCOPY_FALLBACK CONFIG_FORTIFY_SOURCE)

This option checks for obviously wrong memory regions when copying memory to/from the kernel (via copy_to_user() and copy_from_user() functions) by rejecting memory ranges that are larger than the specified heap object, span multiple separately allocated pages, are not on the process stack, or are part of the kernel text.

Hardening Linux:

make linux-xconfig: Security options

→ [*] Harden memory copies between kernel and userspace

[*] Allow usercopy whitelist violations to fallback to object size

[*] Harden common str/mem functions against buffer overflows

File Systems

```
make linux-xconfig: File Systems →
(For each file system you use, make sure extended attribute support is enabled)
     <*> Second extended fs support
         [*]   Ext2 extended attributes
            [*]     Ext2 POSIX Access Control Lists
            [*]     Ext2 Security Labels


     <*> The extended 3 (ext3) file system
         [*]     Ext3 POSIX Access Control Lists
         [*]     Ext3 Security Labels

     <*> The Extended 4 (ext4) file system
         [*]     Ext4 POSIX Access Control Lists
         [*]     Ext4 Security Labels
```

IPv4 protection

More information: <kernel source>/Documentation/networking/ip-sysctl.txt
https://dev.gentoo.org/~swift/docs/security_benchmarks/kernel.html#item-gt-sysctl-ipv4forward

Disable IPv4 Forwarding: If the system is not as a router, `ip_forward` must be disabled:

```
sysctl –w net.ipv4.ip_forward = 0
```

Enable Source Route Verification: if the system is as a router, `rp_filter` should be activated on all interfaces

```
sysctl –w net.ipv4.conf.INT.rp_filter = 1  //INT=eth0, lo, …
```

Disable IP Source Routing: accept_source_route must be disable

```
sysctl –w net.ipv4.conf.INT.accept_source_route= 0        //
                                                INT=eth0, lo, …
```

IPv4 protection

Disable ICMP Redirects: accept_redirect must be disable on all interfaces

```
sysctl –w net.ipv4.conf.INT.accept_redirects = 0   // INT=eth0, lo, …
```

Ignore ICMP Echo Broadcasts: icmp_echo_ignore_broadcasts must be enable

```
sysctl –w net.ipv4.icmp_echo_ignore_broadcasts = 1
```

Ignore ICMP Bogus Error Responses: icmp_ignore_bogus_error_responses must be enable

```
sysctl –w net.ipv4.icmp_ignore_bogus_error_responses = 1
```

Enable Logging of Martians: log_martians should be enable on all interfaces

```
sysctl –w net.ipv4.conf.INT.log_martians=1      //INT=eth0, lo, …
```

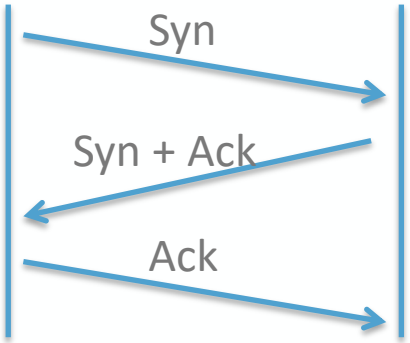
Enable TCP SYN Cookie: tcp_syncookies must be enable

```
sysctl –w net.ipv4.tcp_syncookies=1
```

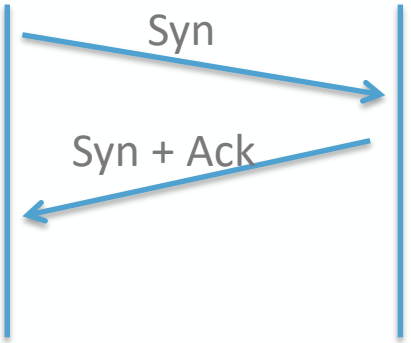
Enable TCP SYN cookie protection

dev.gentoo.org/~swift/docs/security_benchmarks/kernel.html

Normal TCP connection



Half open TCP connection



Enable TCP SYN cookie protection

dev.gentoo.org/~swift/docs/security_benchmarks/kernel.html

Normal TCP/IP networking is open to an attack known as "SYN flooding".

This denial-of-service attack uses half open tcp connections.

The CONFIG_SYN_COOKIES can avoid this attack

make linux-xconfig: Networking support → Networking options → TCP IP Networking → IP TCP Sys Cookies [*]

Read syn cookies configuration

```
cat /proc/sys/net/ipv4/tcp_syncookies  // 0= deactivated, 1=activated
sysctl -n net.ipv4.tcp_syncookies
```

Hardening Linux: Activate sys cookies

```
echo 1 > cat /proc/sys/net/ipv4/tcp_syncookies
sysctl net.ipv4.tcp_syncookies=1
```

Configure kernel parameters

`sysctl –p` command reads the file `/etc/sysctl.conf` and configure kernel parameters

Example:

```
cat /etc/sysctl.conf
   kernel.randomize_va_space=2
   net.ipv4.conf.lo.rp_filter = 1
   net.ipv4.conf.eth0.rp_filter = 1
   net.ipv4.conf.lo.accept_source_route= 0
   net.ipv4.conf.eth0.accept_source_route= 0
   net.ipv4.conf.lo.accept_redirects=0
   net.ipv4.conf.eth0.accept_redirects=0
   net.ipv4.icmp_echo_ignore_broadcasts = 1
   net.ipv4.icmp_ignore_bogus_error_responses = 1
   net.ipv4.conf.lo.log_martians=1
   net.ipv4.conf.eth0.log_martians=1
   net.ipv4.tcp_syncookies=1
```