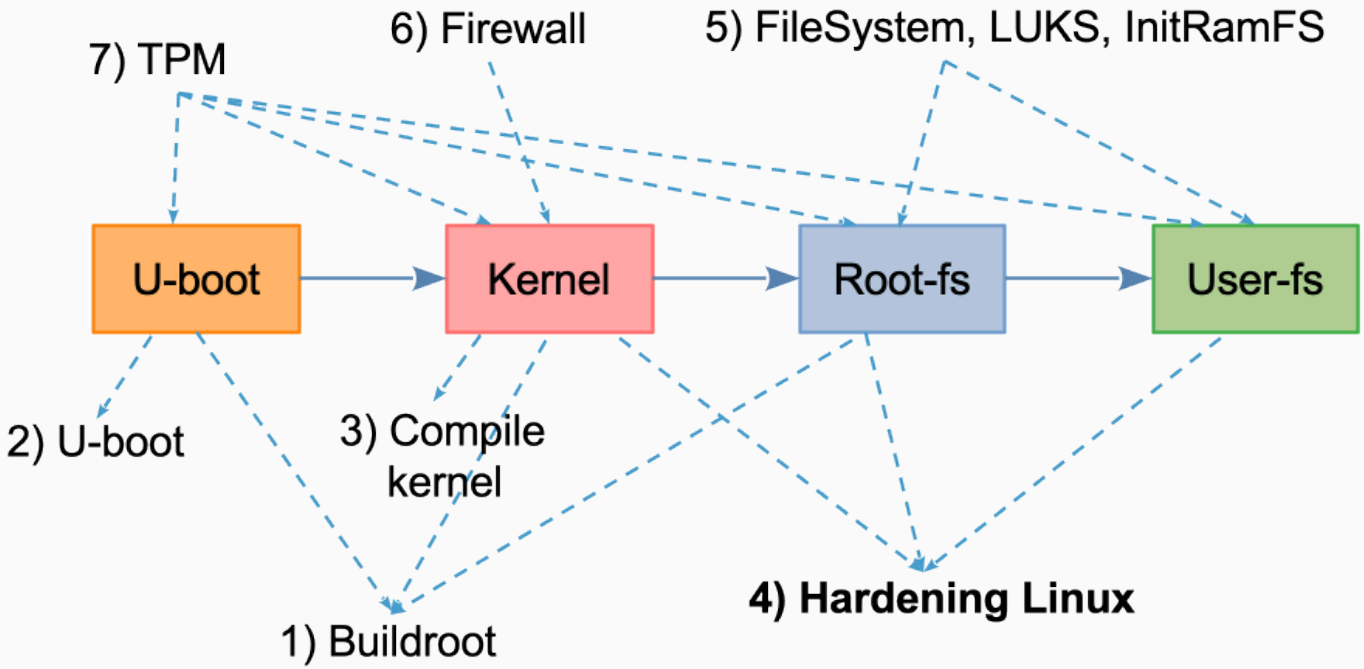
MASTER OF SCIENCE
IN ENGINEERING

MA_SeS - Application Hardening

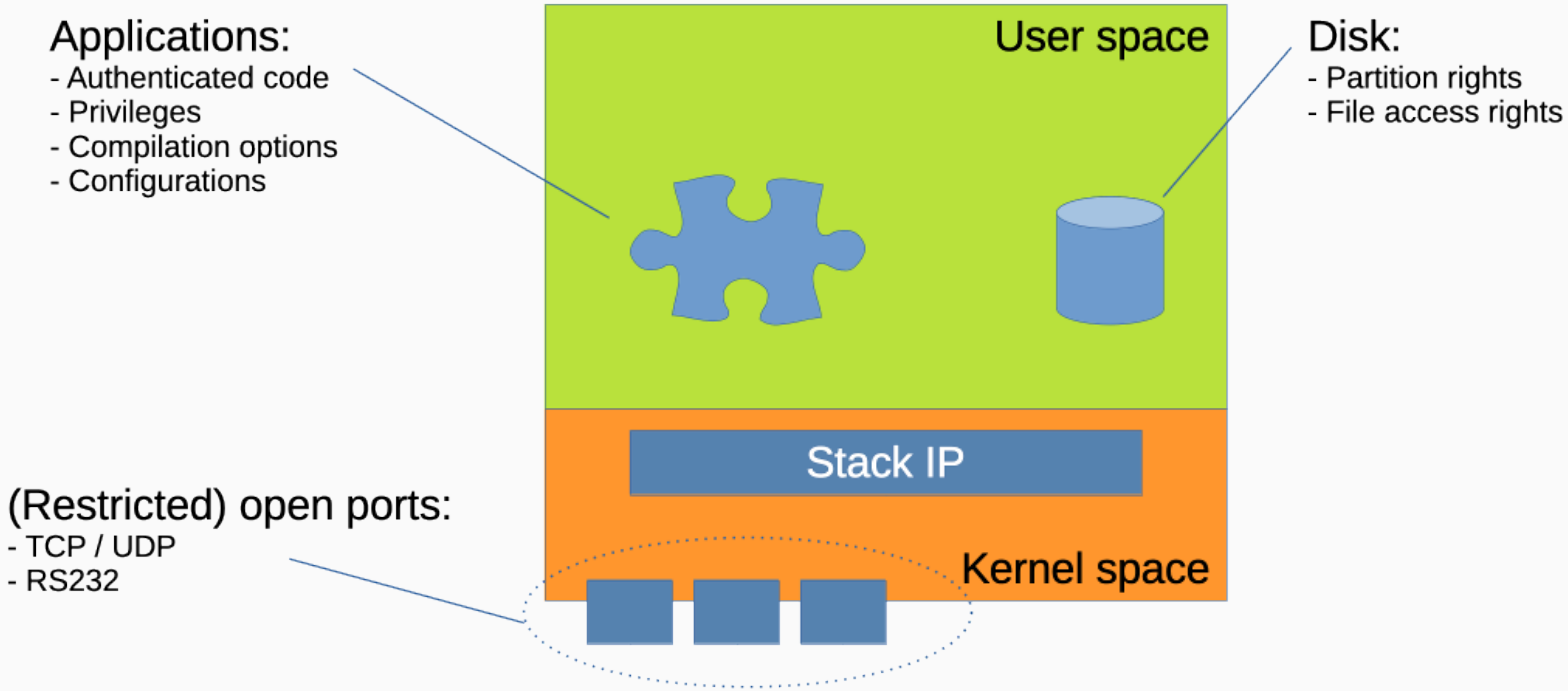
Luca Haab, Michael Mäder, Florent Glück

November 03, 2025

Overview


7) TPM
6) Firewall
5) FileSystem, LUKS, InitRamFS
U-boot
Kernel
Root-fs
User-fs
2) U-boot
3) Compile kernel
4) Hardening Linux
1) Buildroot

Summary


Applications:
- Authenticated code
- Privileges
- Compilation options
- Configurations
User space
Disk:
- Partition rights
- File access rights
Stack IP
(Restricted) open ports:
- TCP / UDP
- RS232
Kernel space

Application Hardening

App hardening - Compilation and linking

- It is possible to use options (already seen in the kernel compilation) below in order to improve the security of an application during the compilation and link phases

```
In the Makefile:
  CFLAGS="-fPIE -fstack-protector-all -D _FORTIFY_SOURCE=2 -ftrapv"
  LDFLAGS="-Wl,-z,now,-z,relro  -z,noexecstack, -pie "


In the command line:
  gcc -Wall -Wextra -z noexecstack -pie -fPIE -fstack-protector-all -Wl,-z,relro,-z,now -O -
D_FORTIFY_SOURCE=2 -ftrapv -o test test.c
```

App hardening - Buffer overflows

Buffer overflows, ret2libc, ROP, ASLR and PIE already seen in the ***Linux kernel configuration and hardening*** *section*

App hardening - noexecstack (1/2)

- Modern systems do not allow execution of instructions stored on the stack. These abbreviations: NX –PaX- DEP, describe the same thing but for different constructors or processors
- **NX bit** means No-eXecute. An operating system and/or hardware which supports the NX bit may mark certain areas of memory as non-executable (stack, heap). The NX bit has different names:
  - Intel: XD bit (eXecute Disable)
  - AMD: Enhanced Virus Protection
  - ARM: XN (eXecute Never)
- The PaX technology can emulate a NX bit (if the NX bit does not exist, ex.: x86 32 bit)
- On Microsoft Windows, the NX bit is called DEP (Data Execution Prevention)
- The execstack option can be used with the Linker flags:

```
LDFLAGS: -z execstack : allow executable stack, disable the NX protection
```

App hardening - noexecstack (1/2)

```
-z noexecstack : not allow executable stack, enable the NX protection
```

See Kernel configuration and hardening section

App hardening - noexecstack (2/2)

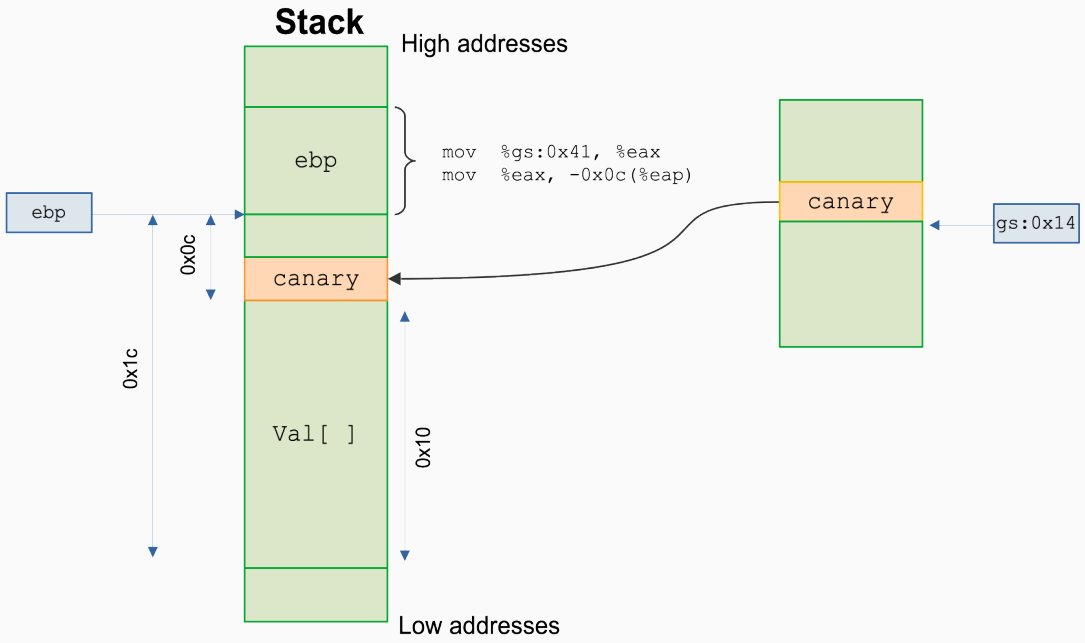
```
$ gcc -Wall -z noexecstack -o test test.c
$ readelf -l test
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x0000000000000714 0x0000000000000714  R E    200000
  LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x000000000000021c 0x0000000000000220  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x00000000000005c0 0x00000000004005c0 0x00000000004005c0
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0  R      1

We can see (highlighted) that the stack is only RW, there is no execution flag
```

See Kernel configuration and hardening section

App hardening - `stack-protector-all`

Stack smashing protection is a way to protect programs from stack buffer overflows by adding random values (**canaries**) between the function's local variables and the saved instruction pointer

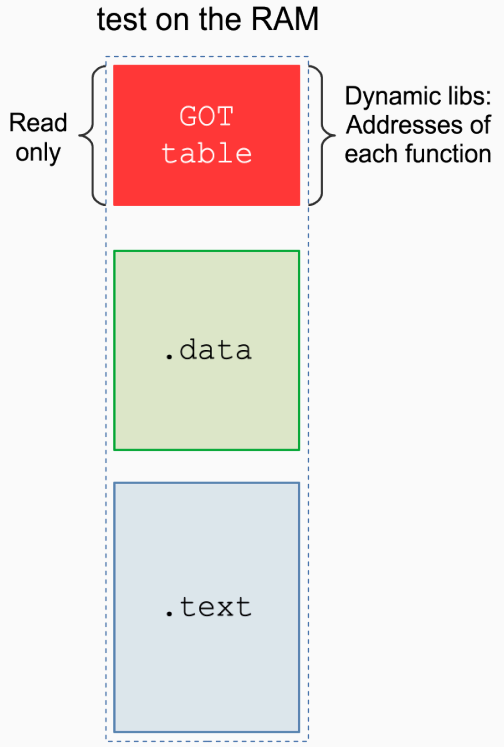
App hardening - stack-protector-all



App hardening - relro and now

- RELRO (RELocation Read-Only) is a security feature that makes some binary sections read-only before starting the program
- It prevents some types of binary exploitation techniques, such as GOT overwrite attacks
- All dynamic symbols must be resolved during program startup ⟶ the GOT (Global Offset Table) table is initialized and made read-only

```
gcc -Wall  -Wl,-z,relro,-z,now -o test test.c
```



App hardening - FORTIFY_SOURCE (1/2)

- FORTIFY_SOURCE macros provide buffer overflow checks for the following functions:

```
memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf,
snprintf, vsnprintf, gets
```

- Example: test.c file:

```
int main(int argc, char **argv) {
char buffer[4];
    strcpy(buffer,argv[1]);
    return (0);
}
```

```
gcc -Wall -O -D_FORTIFY_SOURCE=1 -o test test.c
./test dddd
*** buffer overflow detected ***: ./test terminated
======= Backtrace: =========
/lib64/libc.so.6(__fortify_fail+0x37)[0x3dc090d4f7]
/lib64/libc.so.6[0x3dc090b6b0]
```

App hardening - FORTIFY_SOURCE (2/2)

- If FORTIFY_SOURCE is set to 1, with compiler optimization level 1 (gcc -O1) and above, checks that shouldn't change the behavior of conforming programs are performed
- With FORTIFY_SOURCE set to 2 some more checking is added
  - but some conforming programs might fail!
- Some of the checks can be performed at compile time, the results are in compiler warnings; other checks take place at run time, the results are in a run-time error if the check fails

App hardening - ftrapv

- The ftrapv flag detects integer overflow
- gcc has the option -ftrapv which activates the signal SIGABRT. This signal can be catched
- This option works only with int value (not short or unsigned int)

```
// gcc -Wall -ftrapv -o file file.C -lstdc++
int main(void) {
int a, b, c;
    createCatchSignal ();
    a=200000; b=200000;
    c=a*b;
    printf ("c=%d\n", c);
    return 0;
}
static void createCatchSignal(void) {
struct sigaction act;
    memset (&act, 0, sizeof (act));
    act.sa_handler = catchSignal;
    sigaction (SIGABRT,  &act, 0);
}
static void catchSignal (int signal) {
    printf ("Integer error, signal = %d\n", signal);
    _exit (0);
}
```

App hardening - strnXYZ functionalities (1/2)

- Replace `strcat, strcpy` by `strn`.... Functions
- If you display `man strnpy`, then you get

  `char *strncpy(char *dest, const char *src, size_t n)`
- The `strncpy()` function is similar to `strcpy()`, except that at most `n` bytes of src are copied. **Warning:** If there is no null byte among the first `n` bytes of src, the string placed in dest will not be null-terminated
- If the length of src is less than `n`, `strncpy()` pads the remainder of dest with null bytes
- It is necessary to initialize the buffer to 0 and `n` **must be = `sizeof(buffer)-1`**

```
int main (void)
{
char  buffer[16];
const char *p="AAAAAAAAAAAAAAAAA";  //17bytes
    memset (&buffer[0], 0x0, sizeof(buffer));
    strncpy (&buffer[0], p, sizeof (buffer)-1);
    return (0);
}
```

App hardening - `strnXYZ` functionalities (2/2)

- Some C functions such as `gets()`, `strcpy()`, `strcat()`, `printf()` are known to be insecure because they don't check the size of the destination buffers.
- The following command may be used in order to detect these particular functions in your source code.

```
find . -name "*.c" | xargs egrep 'gets|strcpy|strcat|printf|scanf'
```

end of presentation