Lab 2

1. Discover and implement its license-generation algorithm (keygen).
2. Create a binary patch to disable licensing.

---

1. Dependency Analysis with `ldd`

```
$ ldd ./hack_app
    linux-vdso.so.1 (0x00007ffd42dfe000)
    libcrypto.so.1.1 => /lib/x86_64-linux-gnu/libcrypto.so.1.1
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
    libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
    /lib64/ld-linux-x86-64.so.2 (0x00007f738bede000)
```

- **Notes**: The app links against **OpenSSL** (`libcrypto`) and standard C libraries. No unusual or custom libraries detected.

---

2. Runtime Tracing with `strace`

To locate the license storage, we ran:

```
$ strace -f -e trace=file ./hack_app 2>&1 | tee strace_full.log
```

Then in the `strace.log` file I searched for license-related strings:

The only tracing attribute I found:

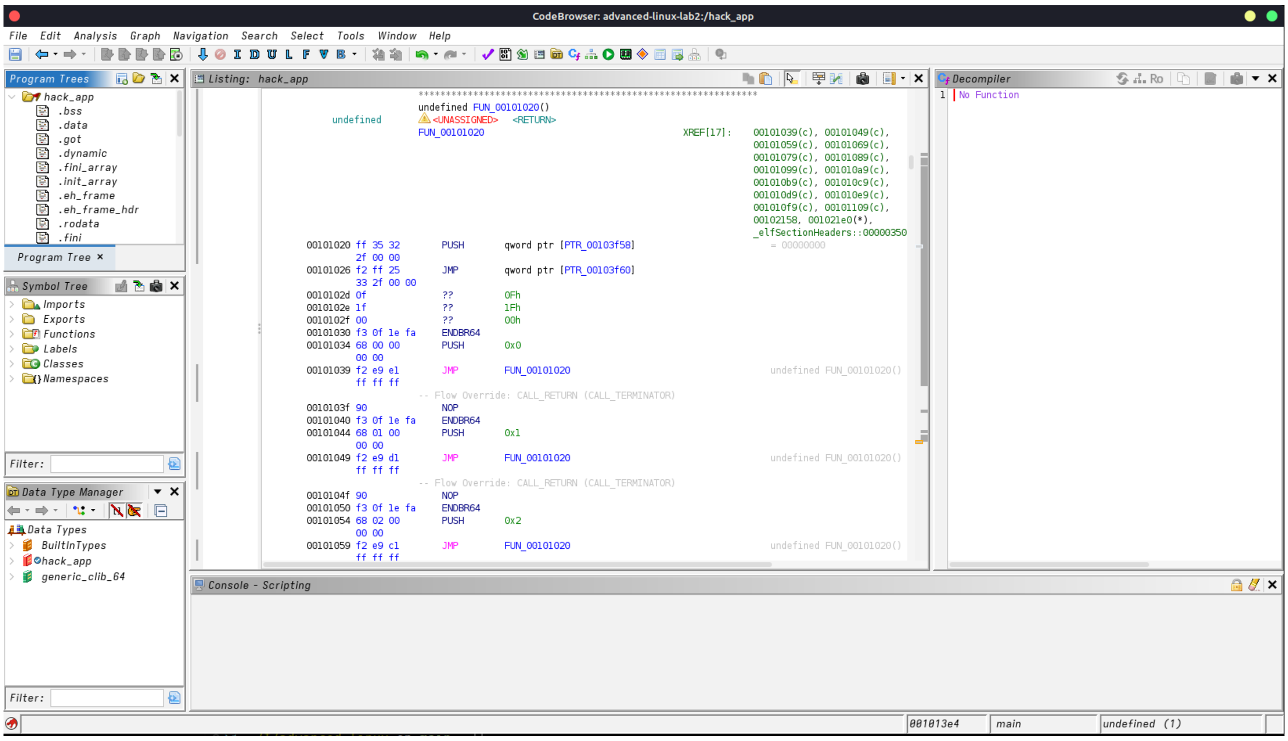
```
getxattr("./hack_app", "user.license", ..., 4096) = -1 ENODATA (No data
available)
```

- The app stores its license in the **`user.license`** extended attribute on its own binary.

---

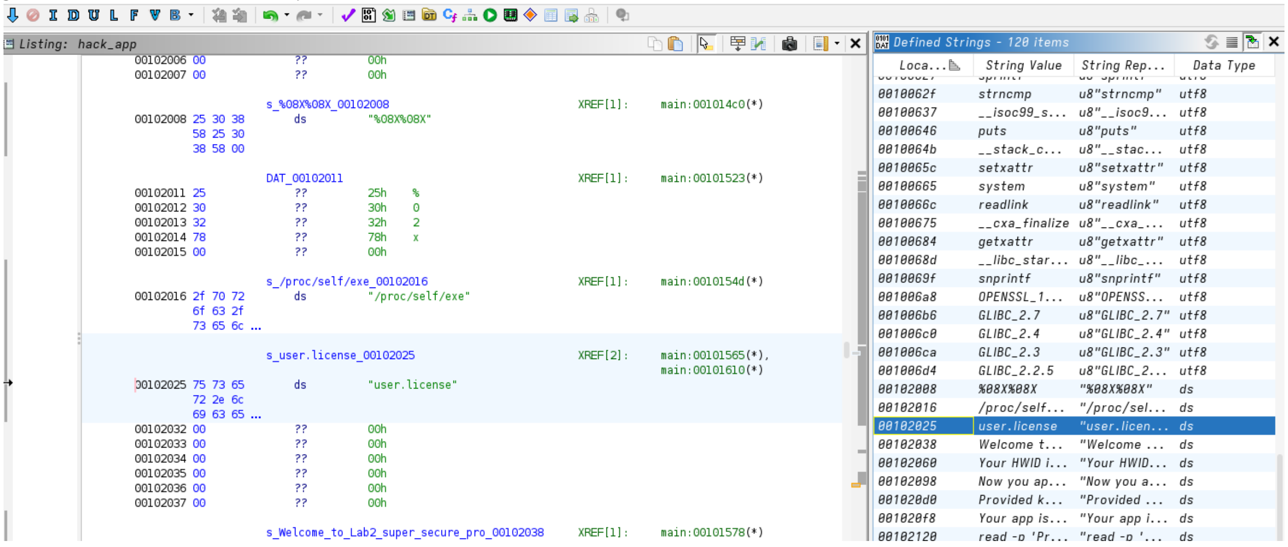
3. Static Analysis in GHIDRA

1. **Import** hack_app into GHIDRA and let auto-analysis finish.



```
                     ****************************************************************
                     undefined FUN_00101020()
       undefined     <UNASSIGNED>   <RETURN>
                     FUN_00101020                      XREF[17]:   00101039(c), 00101049(c),
                                                                   00101059(c), 00101069(c),
                                                                   00101079(c), 00101089(c),
                                                                   00101099(c), 001010a9(c),
                                                                   001010b9(c), 001010c9(c),
                                                                   001010d9(c), 001010e9(c),
                                                                   001010f9(c), 00101109(c),
                                                                   00102158, 001021e0(*),
                                                                   _elfSectionHeaders::00000350
                                                                     = 00000000
00101020 ff 35 32        PUSH       qword ptr [PTR_00103f58]
         2f 00 00
00101026 f2 ff 25        JMP        qword ptr [PTR_00103f60]
         33 2f 00 00
0010102d 0f              ??         0Fh
0010102e 1f              ??         1Fh
0010102f 00              ??         00h
00101030 f3 0f 1e fa     ENDBR64
00101034 68 00 00        PUSH       0x0
         00 00
00101039 f2 e9 e1        JMP        FUN_00101020                         undefined FUN_00101020()
         ff ff ff
                 -- Flow Override: CALL_RETURN (CALL_TERMINATOR)
0010103f 90              NOP
00101040 f3 0f 1e fa     ENDBR64
00101044 68 01 00        PUSH       0x1
         00 00
00101049 f2 e9 d1        JMP        FUN_00101020                         undefined FUN_00101020()
         ff ff ff
                 -- Flow Override: CALL_RETURN (CALL_TERMINATOR)
0010104f 90              NOP
00101050 f3 0f 1e fa     ENDBR64
00101054 68 02 00        PUSH       0x2
         00 00
00101059 f2 e9 c1        JMP        FUN_00101020                         undefined FUN_00101020()
         ff ff ff
```

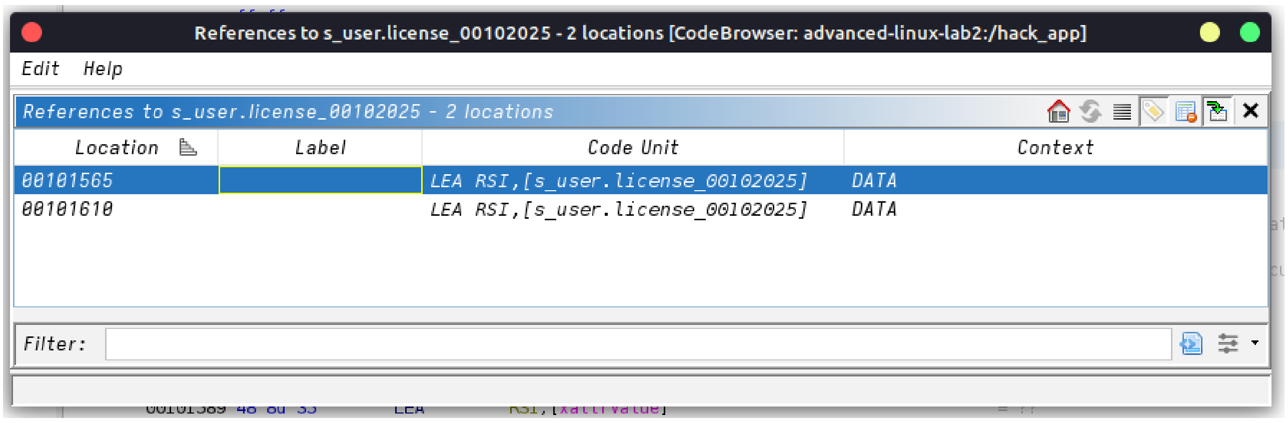
2. **Find** the literal string "user.license" in **Window** → **Defined Strings**.



```
00102006 00              ??         00h
00102007 00              ??         00h

                     s_%08X%08X_00102008                 XREF[1]:    main:001014c0(*)
00102008 25 30 38        ds         "%08X%08X"
         58 25 30
         38 58 00

                     DAT_00102011                        XREF[1]:    main:00101523(*)
00102011 25              ??         25h    %
00102012 30              ??         30h    0
00102013 32              ??         32h    2
00102014 78              ??         78h    x
00102015 00              ??         00h

                     s_/proc/self/exe_00102016           XREF[1]:    main:0010154d(*)
00102016 2f 70 72        ds         "/proc/self/exe"
         6f 63 2f
         73 65 6c ...

                     s_user.license_00102025             XREF[2]:    main:00101565(*),
                                                                     main:00101610(*)
00102025 75 73 65        ds         "user.license"
         72 2e 6c
         69 63 65 ...
00102032 00              ??         00h
00102033 00              ??         00h
00102034 00              ??         00h
00102035 00              ??         00h
00102036 00              ??         00h
00102037 00              ??         00h

                     s_Welcome_to_Lab2_super_secure_pro_00102038   XREF[1]:    main:00101578(*)
```

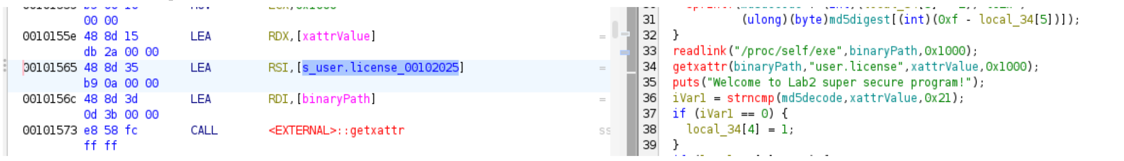
Loca...	String Value	String Rep...	Data Type
0010062f	strncmp	u8"strncmp"	utf8
00100637	__isoc99_s...	u8"__isoc9...	utf8
00100646	puts	u8"puts"	utf8
0010064b	__stack_c...	u8"__stac...	utf8
0010065c	setxattr	u8"setxattr"	utf8
00100665	system	u8"system"	utf8
0010066c	readlink	u8"readlink"	utf8
00100675	__cxa_finalize	u8"__cxa_...	utf8
00100684	getxattr	u8"getxattr"	utf8
0010068d	__libc_star...	u8"__libc_...	utf8
0010069f	snprintf	u8"snprintf"	utf8
001006a8	OPENSSL_1...	u8"OPENSS...	utf8
001006b6	GLIBC_2.7	u8"GLIBC_2.7"	utf8
001006c0	GLIBC_2.4	u8"GLIBC_2.4"	utf8
001006ca	GLIBC_2.3	u8"GLIBC_2.3"	utf8
001006d4	GLIBC_2.2.5	u8"GLIBC_2...	utf8
00102008	%08X%08X	"%08X%08X"	ds
00102016	/proc/self...	"/proc/sel...	ds
00102025	user.license	"user.licen...	ds
00102038	Welcome t...	"Welcome ...	ds
00102060	Your HWID i...	"Your HWID...	ds
00102098	Now you ap...	"Now you a...	ds
001020d0	Provided k...	"Provided ...	ds
001020f8	Your app is...	"Your app i...	ds
00102120	read -p 'Pr...	"read -p '...	ds

3. **Show references** to that string and open the first code reference in main.


References to s_user.license_00102025 - 2 locations [CodeBrowser: advanced-linux-lab2:/hack_app]
Edit Help
References to s_user.license_00102025 - 2 locations
Filter:

Location	Label	Code Unit	Context
00101565		LEA RSI,[s_user.license_00102025]	DATA
00101610		LEA RSI,[s_user.license_00102025]	DATA

4. **Switch** to **Decompiler** view.



```
        00 00
0010155e 48 8d 15    LEA     RDX,[xattrValue]
        db 2a 00 00
00101565 48 8d 35    LEA     RSI,[s_user.license_00102025]
        b9 0a 00 00
0010156c 48 8d 3d    LEA     RDI,[binaryPath]
        0d 3b 00 00
00101573 e8 58 fc    CALL    <EXTERNAL>::getxattr
        ff ff
```

```
            (ulong)(byte)md5digest[(int)(0xf - local_34[5])]);
}
readlink("/proc/self/exe",binaryPath,0x1000);
getxattr(binaryPath,"user.license",xattrValue,0x1000);
puts("Welcome to Lab2 super secure program!");
iVar1 = strncmp(md5decode,xattrValue,0x21);
if (iVar1 == 0) {
  local_34[4] = 1;
}
```

5. **Annotate** the flow:

   - `getxattr(path, "user.license", buf, size)`
   - `if (ret < 0) { prompt user for key }`
   - Call to `__get_cpuid`, byte-swaps, `snprintf(psn, ...)`, `calc_md5`, loop building `md5decode`
   - `strncmp(md5decode, xattrValue, 0x21)` → branch
   - On success: `setxattr(path, "user.license", md5decode, 0x21, 0)`

---

4. Extracting & Implementing the License Algorithm

Based on the decompiled pseudocode:

A. **HWID Computation**

```c
__get_cpuid(1, &eax, NULL, NULL, &edx);
uint32_t hw1 = __builtin_bswap32(eax);
uint32_t hw2 = __builtin_bswap32(edx);
```

B. **PSN Formatting**

```c
char psn[17];
 snprintf(psn, sizeof(psn), "%08X%08X", hw1, hw2);
```

C. **MD5 Calculation**

```c
unsigned char digest[16];
MD5((unsigned char*)psn, 16, digest);
```

D. **License String Construction**

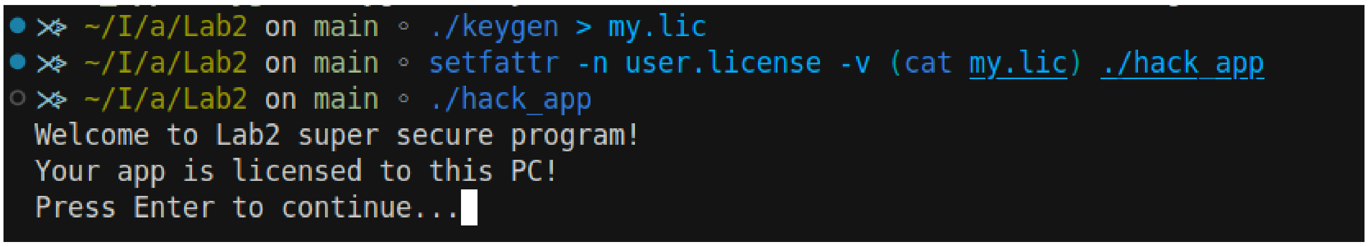
```c
char license[33] = {0};
for (int i = 0; i < 16; i++) {
    sprintf(license + i*2, "%02x", digest[15 - i]);
}
```

4.1 Writing the Keygen (`keygen.c`)

```c
#include <stdio.h>
#include <stdint.h>
#include <cpuid.h>
#include <openssl/md5.h>

int main(void) {
    unsigned int eax, ebx, ecx, edx;
    if (!__get_cpuid(1, &eax, &ebx, &ecx, &edx)) {
        fprintf(stderr, "CPUID not supported.\n");
        return 1;
    }
    uint32_t hw1 = __builtin_bswap32(eax);
    uint32_t hw2 = __builtin_bswap32(edx);
    char psn[17];
    snprintf(psn, sizeof(psn), "%08X%08X", hw1, hw2);
    unsigned char digest[16];
    MD5((unsigned char*)psn, 16, digest);
    char license[33] = {0};
    for (int i = 0; i < 16; i++) {
        sprintf(license + i*2, "%02x", digest[15 - i]);
    }
    printf("%s", license);
    return 0;
}
```

4.2 Compile & Test

```
$ gcc -o keygen keygen.c -lcrypto
$ ./keygen > my.lic
$ setfattr -n user.license -v (cat my.lic) ./hack_app
$ ./hack_app
```

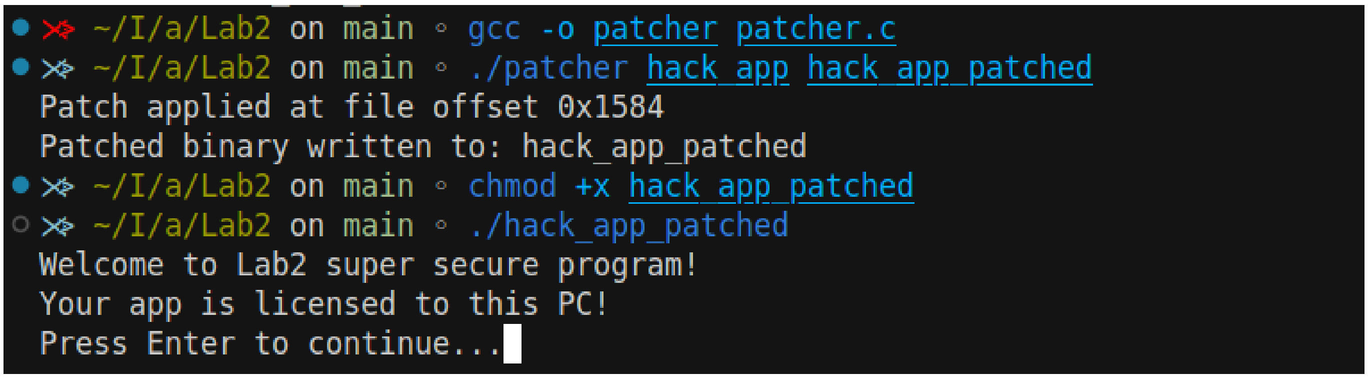

```
● ⋊> ~/I/a/Lab2 on main ◦ ./keygen > my.lic
● ⋊> ~/I/a/Lab2 on main ◦ setfattr -n user.license -v (cat my.lic) ./hack_app
○ ⋊> ~/I/a/Lab2 on main ◦ ./hack_app
  Welcome to Lab2 super secure program!
  Your app is licensed to this PC!
  Press Enter to continue...
```

---

5. Patching

The app does `strncmp(..., 0x21)` (length 33) by loading `0x21` into EDX (`BA 21 00 00 00`). Changing that to `BA 00 00 00 00` makes `strncmp(..., 0)`, which always returns equal, skipping the “wrong key” branch.

To see the code please check the Github Repo



```
● ⋟ ~/I/a/Lab2 on main ◦ gcc -o patcher patcher.c
● ⋟ ~/I/a/Lab2 on main ◦ ./patcher hack_app hack_app_patched
  Patch applied at file offset 0x1584
  Patched binary written to: hack_app_patched
● ⋟ ~/I/a/Lab2 on main ◦ chmod +x hack_app_patched
○ ⋟ ~/I/a/Lab2 on main ◦ ./hack_app_patched
  Welcome to Lab2 super secure program!
  Your app is licensed to this PC!
  Press Enter to continue...
```