
Aktuelle Themen der IT-Sicherheit: RIOT Challenges

WS 22/23: Prof. Dr. Jan Seedorf

Thomas Jakkel (Mat. Nr), Severin Nonenmann (Mat. Nr), Lukas

Reinke (1001213) (Mat. Nr), Lars Weiß (Mat. Nr)

29. Januar 2023



Inhalt

1	Woche 1	2
1.1	Challenge 1: VM installation	2
1.2	Challenge 2: Erste schritte mit RIOT	2
1.2.1	First_test Application	2
1.2.2	Simple Network communication	4
2	Woche 2	5
2.1	Challenge 1	5
2.1.1	Task 01	5
2.2	Challenge 2	6
2.3	Challenge 3	6
2.3.1	Review AES-CBC	6

1 Woche 1

1.1 Challenge 1: VM installation

Das VM setup wird hier nicht genauer beschrieben.

1.2 Challenge 2: Erste schritte mit RIOT

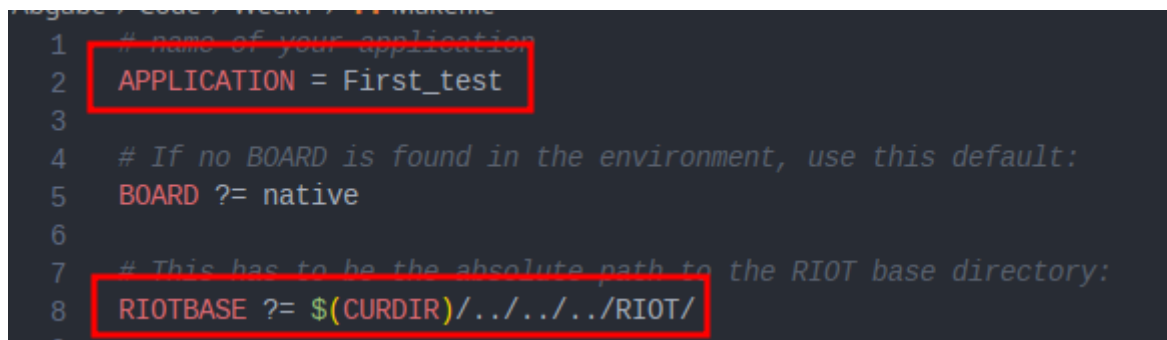
Die Anleitung zum Setup von RIOT OS aus den RIOT Tutorials wurde durchlaufen und ein funktionierender Workspace erstellt.

Wir haben das Setup insofern verändert, dass unser Code in einem separaten Ordner neben dem von GitHub geklonten RIOT Dateien liegt.

1.2.1 First_test Application

Das Ziel ist ein erstes RIOT-OS selber zu kompilieren, mit einer eigen Funktion zu versehen und zu starten.

Im default Makefile müssen zwei Änderungen vorgenommen werden: 1. In der Variable `APPLICATION` der Name der Ausführbaren binary zu setzen 2. Die `RIOTBASE`, dem Pfad zu den Hauptdateien des RIOT-OS, zu setzen.



```
1 # name of your application
2 APPLICATION = First_test
3
4 # If no BOARD is found in the environment, use this default:
5 BOARD ?= native
6
7 # This has to be the absolute path to the RIOT base directory:
8 RIOTBASE ?= $(CURDIR)/../../../../RIOT/
```

Abbildung 1: Einfaches Makefile

Es soll eine shell command geschrieben werden der bei Aufruf einen String ausgibt. Zugrunde liegt eine einfache C Funktion mit einem `printf()` statement:

Des weiteren muss die Funktion in einem Array eingetragen und dieses Array als Quelle für Shell-befehle in der `main` Funktion registriert werden.

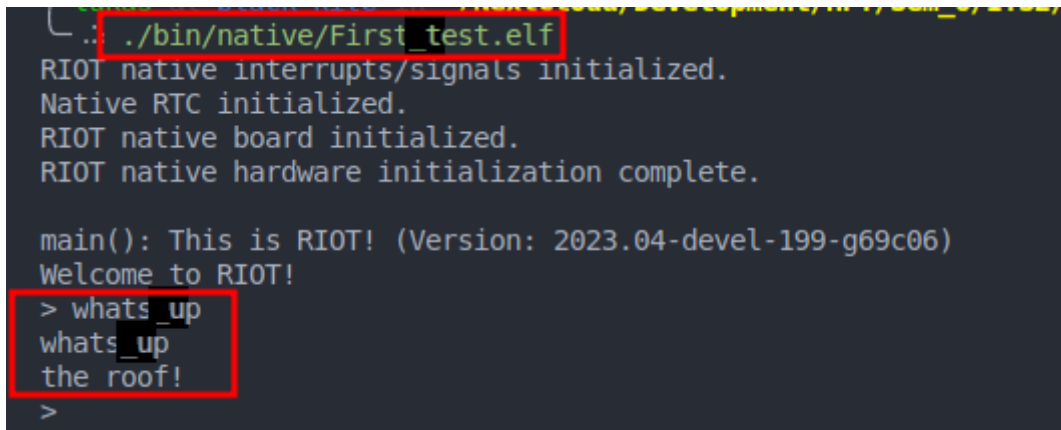
```
35
36 static int whats_up(int argc, char **argv) {
37     (void)argc;
38     (void)argv;
39
40     printf("The roof!\n");
41     return 0;
42 }
43
```

Abbildung 2: Funktion Whats_up

```
43
44 const shell_command_t shell_commands[] = {
45     {"whats_up", "prints the roof", whats_up},
46     { NULL, NULL, NULL}
47 };
48
49 int main(void)
50 {
51     #ifdef MODULE_NETIF
52     gnrc_netreg_entry_t dump = GNRC_NETREG_ENTRY_INIT_PID(GNRC_NETREG_DEMUX_CTX_ALL,
53     gnrc_pktdump_pid);
54     gnrc_netreg_register(GNRC_NETTYPE_UNDEF, &dump);
55     #endif
56
57     (void) puts("Welcome to RIOT!");
58
59     char line_buf[SHELL_DEFAULT_BUFSIZE];
60     shell_run(shell_commands, line_buf, SHELL_DEFAULT_BUFSIZE);
61
62     return 0;
63 }
64
```

Abbildung 3: Shell Kommando Registrieren

Nun kann mithilfe des `make`-Kommandos ein build gestartet werden und die resultierende Binary mit dem Namen **First_test.elf** ausgeführt werden. In der RIOT Shell kann nun der Befehl `whats_up` ausgeführt werden.



```
./bin/native/First_test.elf
RIOT native interrupts/signals initialized.
Native RTC initialized.
RIOT native board initialized.
RIOT native hardware initialization complete.

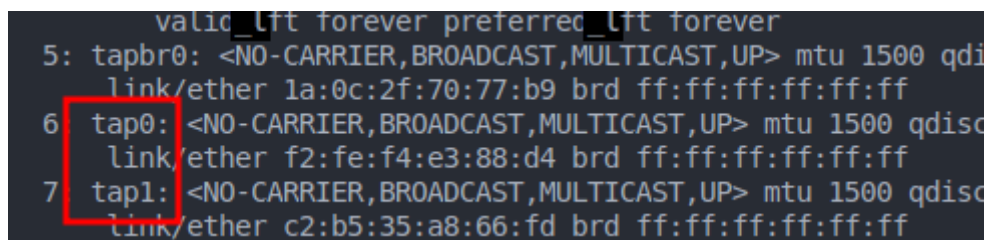
main(): This is RIOT! (Version: 2023.04-devel-199-g69c06)
Welcome to RIOT!
> whats_up
whats_up
the roof!
>
```

Abbildung 4: `whats_up` Befehl in RIOT shell

1.2.2 Simple Network communication

Das Ziel dieser Challenge war, zwei RIOT-OS Instanzen über Netzwerk kommunizieren zu lassen.

Das im RIOT Repo mitgelieferte Script `tapsetup` kann genutzt werden um in der Linux Umgebung zwei interfaces (`tap0` und `tap1`) anzulegen.



```
valid_lft forever preferred_lft forever
5: tapbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
   link/ether 1a:0c:2f:70:77:b9 brd ff:ff:ff:ff:ff:ff
6: tap0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
   link/ether f2:fe:f4:e3:88:d4 brd ff:ff:ff:ff:ff:ff
7: tap1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
   link/ether c2:b5:35:a8:66:fd brd ff:ff:ff:ff:ff:ff
```

Abbildung 5: Zwei virtuelle Interfaces mit verbindender “bridge”

Wird nun eine RIOT-OS Instanz mit dem Zusatz `PORT=tap0` ist das Interface `tap0` Verbunden und kann intern mit dem Befehl `ifconfig` gefunden werden. Neben dem Interface wird die Hardware-Adresse auf der das Interface später angesprochen werden kann angezeigt.

Nun kann mit Hilfe des Befehls `txtsnd 4 C2:B5:35:A8:66:FE hello` eine Nachricht an ein anderes Interface gesendet werden. Der Befehl beinhaltet: 1. Die Interface Nummer auf der gesendet werden soll 2. Die Hardware Adresse des Ziels 3. Die Nachricht

```
...: sudo ./bin/native/First_test.elf PORT=tap1
RIOT native interrupts/signals initialized.
Native RTC initialized.
RIOT native board initialized.
RIOT native hardware initialization complete.

main(): This is RIOT! (Version: 2023.04-devel-199-g69c06)
Welcome to RIOT!
> ifconfig
ifconfig
Iface 4 HWaddr: B6:46:4F:29:49:0B
L2-PDU:1500 Source address length: 6
```

Abbildung 6: tap1 in RIOT-OS

Auf dem zweiten Instanz kann die gesendete Nachricht nun in Hexadezimaler Form empfangen werden.

```
> PKTDUMP: data received:
~ SNIP 0 - size: 5 byte, type: NETTYPE_UNDEF (0)
00000000 68 65 6C 6C 6F
~ SNIP 1 - size: 20 byte, type: NETTYPE_NETIF (-1)
if_pid: 4 rssi: -32768 lqi: 0
flags: 0x0
src_l2addr: F2:FE:F4:E3:88:D5
dst_l2addr: C2:B5:35:A8:66:FE
~ PKT - 2 snips, total size: 25 byte
```

Abbildung 7: Empfangene Nachricht

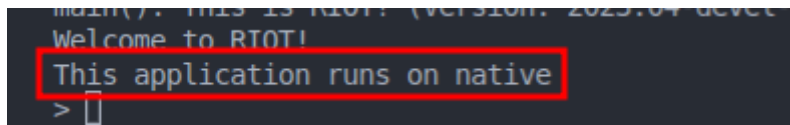
2 Woche 2

2.1 Challenge 1

Das Ziel ist die ersten vier Tutorials von RIOT durchzuarbeiten

2.1.1 Task 01

Einfügen der Code-Zeile `printf("This application runs on %s\n", RIOT_BOARD);` gibt den Hardware Typ aus für den RIOT Kompiliert wurde.



```
main(): THIS IS RIOT! (VERSION: 2023.04-devel)
Welcome to RIOT!
This application runs on native
> []
```

Abbildung 8: Ausgabe der neuen codezeile

2.2 Challenge 2

Lorem Ipsum

2.3 Challenge 3

2.3.1 Review AES-CBC

Im AES-CBC