DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

An Approach for Home Routers to Securely Erase Sensitive Data

Nicolas Beneš

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

An Approach for Home Routers to Securely Erase Sensitive Data

Ein Lösungsansatz für Heimrouter zum sicheren Löschen empfindlicher Daten

Author:	Nicolas Beneš
Supervisor:	Christian Grothoff, PhD (UCLA)
Advisor:	Christian Grothoff, PhD (UCLA)
Date:	October 15, 2014

I assure the single handed composition of this Bachelor Thesis only supported by declared resources.

München, October 15, 2014 Nicolas Beneš

Acknowledgements

First of all, I thank Christian Grothoff for his continuous encouragement and patience throughout the creation of this Bachelor Thesis. His guidance and expertise helped me to orient myself in the security domain to which I had no contact before.

Furthermore, I like to thank Jacob Appelbaum who proposed the initial idea for the topic of this thesis. It was a lot of fun to work on this project.

Also, I like to thank Thomas Grübler for his willingness to answer my questions regarding circuit design and layout, as well as Martin Saß and Andreas Hauptner from the Department of Physics for providing me access to a water-jet vacuum pump, and the Institute for Cognitive Systems for allowing me to use their reflow oven.

Abstract

Home routers are always-on low power embedded systems and part of the Internet infrastructure. In addition to the basic router functionality, they can be used to operate sensitive personal services, such as for private web and email servers, secure peer-to-peer networking services like GNUnet and Tor, and encrypted network file system services. These services naturally involve cryptographic operations with the cleartext keys being stored in RAM. This makes router devices possible targets to physical attacks by home intruders. Attacks include interception of unprotected data on bus wires, alteration of firmware through exposed JTAG headers, or recovery of cryptographic keys through the cold boot attack.

This thesis presents *Panic!*, a combination of open hardware design and free software to detect physical integrity attacks and to react by securely erasing cryptographic keys and other sensitive data from memory. To improve auditability and to allow cheap reproduction, the components of *Panic!* are kept simple in terms of conceptual design and lines of code.

First, the motivation to use home routers for services besides routing and the need to protect their physical integrity is discussed. Second, the idea and functionality of the *Panic!* system is introduced and the high-level interactions between its components explained. Third, the software components to be run on the router are described. Fourth, the requirements of the measurement circuit are declared and a prototype is presented. Fifth, some characteristics of pressurized environments are discussed and the difficulties for finding adequate containments are explained. Finally, an outlook to tasks left for the future is given.

Contents

1. Introduction

1.1. Home Routers in Peer-to-Peer Networks

Home routers are cheap always-on low power devices that usually connect multiple devices in a LAN to the Internet. As technology evolves, they often have more compute power and resources than actually needed for their primary purpose, i.e. routing. Therefore, many devices provide additional functions to be used as private file, web, or email server. Systems based on OpenWrt[1] firmware and similar even allow to run any software that can be compiled for the target platform and executed in the limits of the hardware [3].

The special cultural and legal protection framework offered to one's home [17] make home routers an attractive location for sensitive private information. Because home routers also often have a network interface that is not subject to network address translation (NAT), their location on the network makes these devices suitable for use in privacy, anonymity, and censorship resistant peer-to-peer networks, such as GNUnet[2] and Tor[3]. A home router may therefore be useful as a cheap Tor bridge or to operate a hidden service [27]. Additionally, these devices may also be used as transparent proxies [27] to the network, omitting the need to install and configure the peer-to-peer software on every local client.

1.2. Risk of Physical-Access Attacks on Home Routers

For political activists and journalists using home routers for sensitive information, this equipment may become target to physical attacks by adversaries not respecting the sancity of the home. However, typical home routers lack meaningful protection against physical attacks. Making such protection available for the before mentioned cases is important, as the use of effective security measures can be essential up to protecting an individual's life and limb [9, 29:58–31:53].

[1] `https://openwrt.org`
[2] `https://gnunet.org/`
[3] `https://www.torproject.org/`

For this work, we assume that the software on the home router is secure and focus on attacks that make use of physical access to the device hardware, of which some examples are given below.

1.2.1. Interception of Data on Bus Wires

Historically [29], the CPU and the primary memory are physically located in distinct devices, for example several DRAM ICs are soldered on a SO-DIMM module which is attached to a connector on a PCB. The socket is then connected to the memory bus of the CPU. Depending on the system, it is also common that the ICs are soldered directly on the PCB without a connector.

In either case, the electrical connection between CPU and memory is vulnerable to interception of transmitted data, for example by attaching a logic analyser to the exposed SO-DIMM connector or PCB traces shown in figure 1.1. Thus, it is easy for an attacker to read all exchanged data off the bus, including clear text encryption keys and other sensitive information.

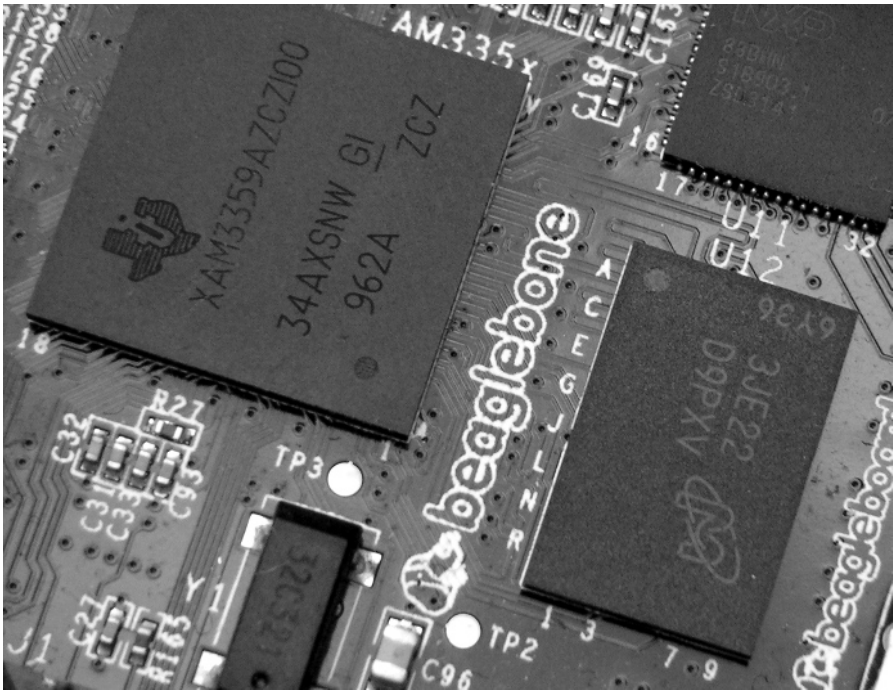


Figure 1.1.: PCB traces between CPU and DRAM IC on a Beaglebone Black.

1.2.2. Alteration of Firmware through the JTAG Interface

The IEEE Joint Test Action Group defined the standard IEEE 1149.1 [1], which is commonly referred to as JTAG. The JTAG interface is intended for testing and programming of integrated circuits [2]. Consequently, a JTAG pin header is exposed on many router PCBs, similar to the one depicted in figure 1.2.

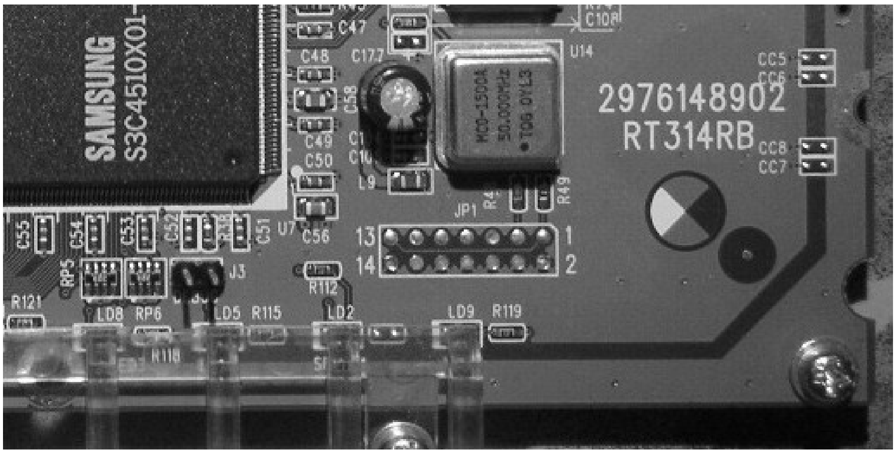


Figure 1.2.: Exposed 14-pin JTAG interface marked JP1 on a Netgear RT314 router.

Just like a legitimate tester, an attacker can use the JTAG interface to read and program the flash IC [2] used for booting the device and install a rootkit or other malware [13].

1.2.3. Memory Recovery through the Cold Boot Attack

As third example, the cold boot attack [18] allows an attacker to recover encryption keys stored in memory in the clear. The attack is based on the remanence effect of DRAM, i.e. the contents of DRAM cells do not decay immediately after power-off, rather gradually during the next couple of minutes. If an attacker boots prepared software from an external device or moves the memory module to another computer, it is possible to image the memory and recover cryptographic keys. Moreover, an attacker can slow down the decay process to the range of hours and longer by cooling the DRAM modules, for example by using cooling spray or liquid nitrogen.

1.3. Contribution of the Thesis

This thesis presents *Panic!*, a combination of an open hardware design and free software that allows users to physically secure suitable home routers (in particular models sufficiently similar to the Beaglebone Black using a Linux kernel) against a wide range of physical attacks. The basic idea is to trigger a customizable panic logic that erases sensitive information whenever an attempt to compromise the physical integrity of the system is detected.

2. The Panic! System

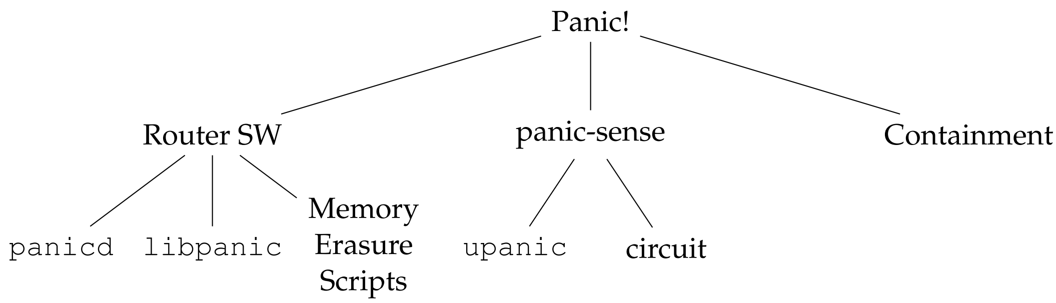


Figure 2.1.: *Panic!* component breakdown structure.

As depicted in figure 2.1, *Panic!* can be split into three mostly independent components:

1. Software to be run on the router, consisting of
   a) the system daemon `panicd` to monitor a GPIO pin for a trigger signal and inform other processes about that trigger,
   b) the library `libpanic` to be loaded into individual processes to erase their memory in a prioritised way in case of a trigger,
   c) and several scripts adapted from Tails [12] to kill the system and finally erase all of the system memory;
2. A panic-sense circuit to detect an attack via several sensors which are readout and evaluated by the `upanic` firmware on an Atmel AVR XMEGA microcontroller;
3. An air-tight and pressurized containment to provide a protected environment for the router and panic-sense circuit.

Figure 2.2 illustrates the interconnections of these components: the panic-sense circuit and the router are connected and screwed together, then, the module is put inside the containment. The containment is pressurized through a valve and represents the physical system boundary.

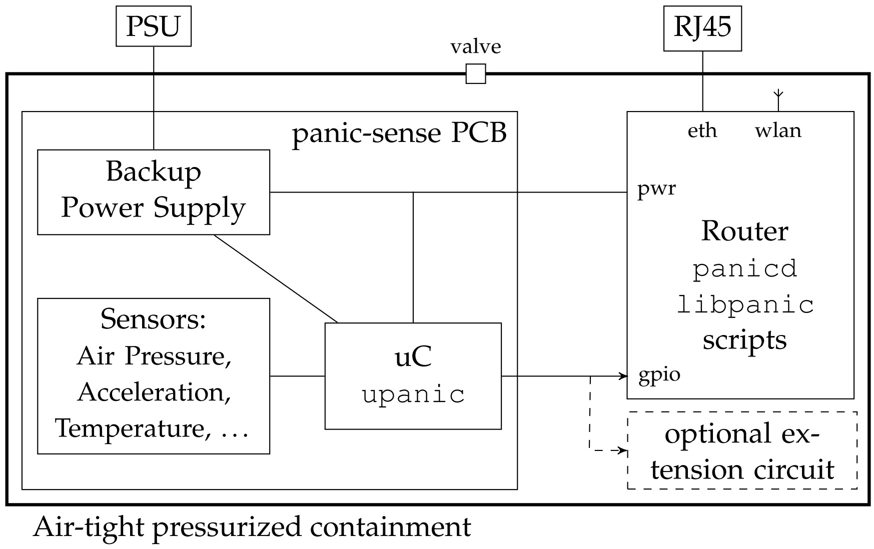


Figure 2.2.: Physical structure of the *Panic!* system.

From a software perspective, the `panicd` daemon monitors a GPIO pin of the router for a falling edge. If the the pin gets triggered, processes that use `libpanic` immediately erase their memory. After a short timeout, additional scripts get triggered and cause erasure for the entire memory and halt of the system.

2.1. Router Platform

The router software of *Panic!* is intended to be run on Linux systems, such as Debian GNU/Linux for ARM and OpenWrt for MIPS hardware, and uses non-portable Linux specific features. Limiting the scope to ARM and MIPS platforms already covers a large number [4] of available commercial off-the-shelf routers and cheap single-board embedded Linux computers.

For testing and as demonstration platform, a Beaglebone Black[1] (BBB) single-board computer is used. The BBB as depicted in figure 2.3 provides a 1 GHz Cortex-A8 ARM processor with 512 MB RAM and runs Debian GNU/Linux.

[1] `https://elinux.org/Beagleboard:BeagleBoneBlack`

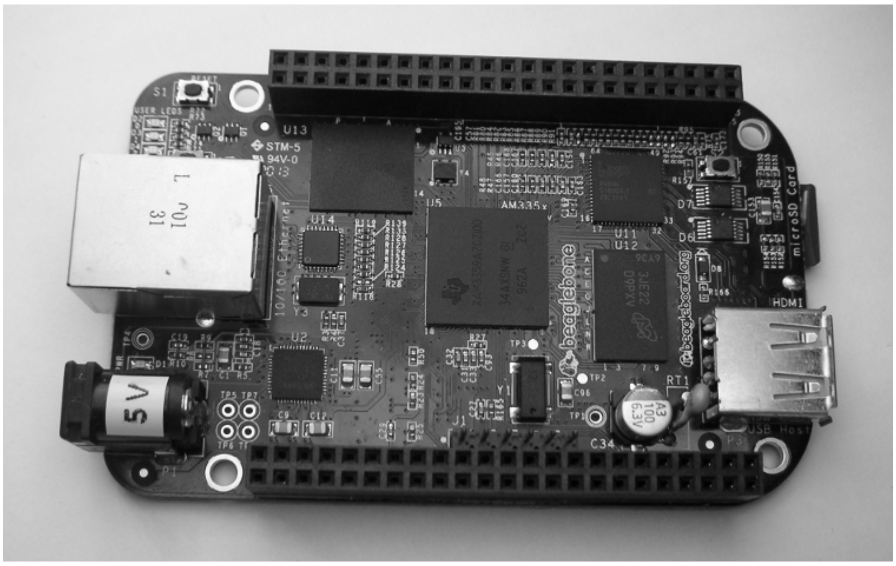


Figure 2.3.: Beaglebone Black rev. A5C.

3. Router Software

3.1. System Daemon panicd

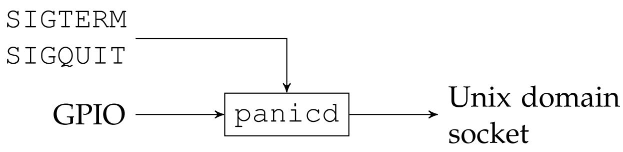


Figure 3.1.: Interfaces of `panicd`.

The system daemon `panicd` monitors a general-purpose input/output (GPIO) pin of the router's processor for a logic high-to-low transition (falling edge), or the `SIGTERM` and `SIGQUIT` signals to trigger notification of client processes. As notification mechanism, `panicd` provides a Unix domain socket to which these processes can connect to.

3.1.1. Usage

```
panicd --gpio GPIO_NUM [--socket PATH][--daemon][--verbose]
```

Listing 3.1: `panicd` command line parameters

As shown in listing 3.1, `panicd` has four command line parameters:

- `--gpio GPIO_NUM` is mandatory to select the GPIO pin that shall be monitored for the trigger. The option takes the pin's logic number as argument to export it to the `/sys/class/gpio/` tree and configure it as edge sensitive input;
- `--socket PATH` can be used to override the default path of the Unix domain socket. If this option is given, it is the users responsibility to properly set the permissions for other processes to access the socket file and the directory.

  If the option is omitted, the abstract Unix domain socket `\0panicd` is created instead, allowing access from all processes without the need to configure permissions;

- `--daemon` instructs `panicd` to run as background process. This option is useful, if the user wants to use `panicd` stand-alone, for example outside of shell scripts.

  The option can be omitted to make `panicd` stay in foreground, for instance when used together with consecutive commands within a shell script;
- `--verbose` increases the amount of diagnostic logging messages that are sent to syslog.

The `panicd` process can be triggered and terminated by the following three sources:

- a falling edge on the selected GPIO pin,
- the reception of a `SIGTERM` signal,
- the reception of a `SIGQUIT` signal.

Depending on the trigger source, either `0` (GPIO pin, `SIGQUIT`) or `1` (`SIGTERM`) is returned as exit status. This permits the use of `panicd` in a script concatenating other commands, for example as given for the scenario of maintenance in listing 3.2.

However, processes using the Unix domain socket through `libpanic` will be killed regardless of the trigger source and therefore should be terminated by the user prior to `panicd`.

```
panicd --gpio 7 && echo "HELP!" | mail emergency@example.com &
# The email is sent, if the GPIO pin is triggered
# or if the user sends SIGQUIT.
# The email is NOT sent, if the user sends SIGTERM.
shutdown -h now
# shutdown sends processes the SIGTERM signal.
```

Listing 3.2: Terminating `panicd` for maintenance without triggering emergency actions.

3.1.2. Implementation

```
int main (int argc, char **argv)
{
    parse_cmd_line (...);
    setup_gpio (pinnum);
    setup_socket (panicd_socket);
```

```
    if (is_daemonize)
        daemon (...);
    cpid = fork ();
    if (0 == cpid) {        /* child */
        while (1)
            accept (...);
    } else {                /* parent */
        check_gpio (...);   /* blocks until trigger or signal */

        kill (cpid, SIGKILL);
        wait (...);
        clean_up_socket (...);
        clean_up_gpio (...);

        nanosleep (500 ms);
    }
    exit (...);
}
```

Listing 3.3: `panicd` implementation (simplified).

As shown in listing 3.3, `panicd` initializes and then forks into a parent and a child process. The child process enters an endless loop accepting connections and blocks if none are pending. As described in section 3.2, clients call `read()` on the socket, but never receive a reply from `panicd`. Instead, the function fails in case of a trigger.

Similarly, the parent process calls `ppoll()` and `read()` on the file descriptor of the GPIO pin. As the pin is configured as interrupt input, the process is blocked as long as no new events occur, thus, both the parent and the child process are blocked during normal (non-triggered) operation.

In case one of the triggers is received, the parent process unblocks and kills the child, thus, causing the operating system to close the connection file descriptors which in turn force the `libpanic`-side `read()` to fail.

Finally, `panicd` sleeps for 500 ms before exiting. This gives processes using the `libpanic` library time to erase their memory and delays the execution of consecutive commands if `panicd` is used in shell scripts.

GPIO Pin

As already described, `panicd` uses an edge sensitive GPIO pin to interface the router and external measurement circuit.

Compared to other hardware interfaces present on a generic router, like USB, UART, or Ethernet, the use of one GPIO pin provides a physically and logically simple mechanism to transmit a binary status to the software. Moreover, using the pin in interrupt configuration avoids busy waiting and safes CPU time.

In general, at least one GPIO pin should be available on a router, for example a switch to enable or disable the wireless output and several status LEDs. Consequently, this hardware requirement is easy to be met by many routers.

Unix Domain Socket

The notification mechanism of `panicd` relies on the Unix domain socket [21], which provides an efficient interface for local interprocess communication. By using the Linux specific abstract Unix domain socket, the user does not need to administer file and directory permissions. However, if the permissions are needed, for example to isolate multiple program groups, the normal non-abstract Unix domain socket can still be used.

3.2. Library libpanic

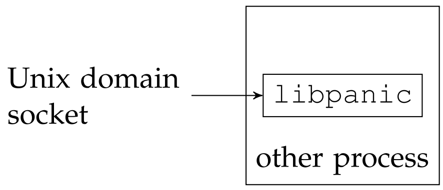


Figure 3.2.: Interfaces of `libpanic`.

The `libpanic` library interfaces the Unix domain socket from `panicd` and provides generic memory erasure functionality. It can be loaded using `LD_PRELOAD`, or linked during build time into almost any program. In case `panicd` receives a trigger, the signal is forwarded to the library, which suspends the current program execution. By default, `libpanic` then overwrites all writeable pages mapped into the virtual address space of the process, except the stack of the thread executing `libpanic` code.

3.2.1. Usage

```
LD_PRELOAD=libpanic.so tor --defaults-torrc /to/torrc
```

Listing 3.4: Example for loading `libpanic` into Tor.

For most programs, it should be sufficient to use `LD_PRELOAD` as in listing 3.4 to make use of `libpanic`. This however is not possible for programs which drop environment variables at some point during execution. In that case, it is necessary to link `libpanic` directly into the executable; see section 3.2.6 for an example using the OpenSSH daemon[1].

Environment Variables

As the library initializes before the actual `main()` function is executed, it is not possible to pass command line options to the library. Instead, the following environment variables are used to set settings diverging from the default:

- `LIBPANIC_SOCKET_PATH=PATH` can be used to override the default path where to look for `panicd`'s Unix domain socket. The library immediately terminates the process if it cannot access the specified socket, for example if `panicd` is not started, if the socket is not present at the given location, or if it is not accessible due to lack of permissions of the current process.

  If the option is omitted, the abstract Unix domain socket `\0panicd` is used instead;

- Setting `LIBPANIC_DEBUG` increases the amount of diagnostic logging messages that are sent to syslog. In case of a trigger, this includes the virtual memory mapping table prior to erasure.

API

The library provides a single API function as shown in listing 3.5, which can be used to assign a callback function called prior to the built-in memory erasure routine. It allows to adjust the behaviour in case of a trigger, for example to provide a custom function to erase memory known to contain sensitive data or to shred hidden service identity key files. It has to be noted though, that the actions of all processes using `libpanic` must fit in `panicd`'s 500 ms time frame.

[1] `http://www.openssh.com/index.html`

```
void panic_set_callback (int (*const cb) (void));
```

Listing 3.5: API of `libpanic`.

Moreover, the callback function can be used to entirely disable `libpanic`'s built-in memory erasure functionality by returning a non-zero value. In turn, built-in memory erasure is enabled if `0` is returned.

It is possible to unset a previously set callback handler by passing `NULL` as argument to the API function.

3.2.2. Implementation

The main functionality of `libpanic` runs as separate thread within the process the library is loaded into. As this is an essential design decision, see section 3.2.3 for other approaches that were considered but had to be rejected.

Library Initialization, Exiting, and Propagation

```
static void panic_init (void) __attribute__ ((constructor));
static void panic_exit (void) __attribute__ ((destructor));
static void atfork_child_handler (void);

static void
panic_init (void)
{
  int (*orig_create) (...);
  int (*orig_atfork) (...);
  orig_create = (void *) dlsym (RTLD_NEXT, "pthread_create");
  orig_atfork = (void *) dlsym (RTLD_NEXT, "pthread_atfork");
  ...
  main_thread = pthread_self ();
  mapsfilefd = open ("/proc/self/maps", O_RDONLY | O_CLOEXEC);
  ...
  if (-1 == sockfd)
    {
      const char *panicd_sock_path = ...;
      ...
```

```
      sockfd = socket (AF_UNIX, SOCK_STREAM | SOCK_CLOEXEC, 0);
      ...
      connect (sockfd, (struct sockaddr *) &panicd_addr,
               sizeof (struct sockaddr_un));
      ...
    }
  orig_atfork (NULL, NULL, &atfork_child_handler);
  orig_create (&panic_thread, NULL, &panic_handler, NULL);
  ...
}

static void
panic_exit (void)
{
  int (*orig_cancel) (pthread_t);
  orig_cancel  = (void *) dlsym (RTLD_NEXT, "pthread_cancel");
  orig_cancel (panic_thread);
  clean_up ();
}

static void
atfork_child_handler (void)
{
  clean_up ();
  panic_init ();
}
```

Listing 3.6: Functions to initialize, exit, and propagate `libpanic` (simplified).

As can be seen in listing 3.6, initialization of `libpanic` takes place before entering `main()` by using the `constructor` function attribute[2] for `panic_init()`. Similarly, the clean-up function `panic_exit()` is called after completion of `main()` using the `destructor` attribute.

During initialization, the file `/proc/self/maps` "containing the currently mapped memory regions" [22] is opened. Following a trigger, it is used to determine the virtual memory regions to be erased.

[2]https://gcc.gnu.org/onlinedocs/gcc/Function-Attributes.html

If the library is run for the first time or after a call to `exec()`, it connects to `panicd`'s Unix domain socket; however, an existing connection is reused in case of `fork()`. Allocating the file descriptors for `maps` file and socket as early as initialization allows immediate exit on error, and prevents failures from open file descriptor limitations and other similar failures.

At the end of initialization, a child handler is assigned through `pthread_atfork()`. In case a program calls `fork()`, the `panic_init()` function is not executed automatically, thus, assigning the handler allows to explicitly initialize `libpanic` in the child process and its propagation.

Finally, the actual functionality of `libpanic` is started in a new thread.

Wrapper Functions

The `libpanic` library hides several functions by it's own wrappers to provide conflict free operation:

- `pthread_create()` adds a freshly created thread to an internal list. The list is used to determine the threads that need to be stopped in case of a trigger. Additionally, it wraps the start routine the user specified;
- `wrapped_start_routine()` is a wrapper to the user's start routine passed to `pthread_create()` and removes the thread from the library internal list on completion. It makes use of `pthread_cleanup_push()`/`-pop()` functions to remove the item from the list even if a thread calls `pthread_exit()`.
- `pthread_cancel()` kills the given thread and removes it from the thread list;
- `pthread_atfork()` handles the case of a program calling `pthread_atfork()` with `NULL` for the child handler parameter. Doing so without the wrapper removes all registered functions including the `atfork_child_handler()` used to propagate `libpanic` to forked processes. In this case, however, the wrapper re-adds the `atfork_child_handler()`;
- `close()` is wrapped to forbid other threads to close the file descriptors for the `maps` file and the Unix socket. If that happens, the function returns a value indicating success, but actually does nothing. An example of a program with such a behaviour is the OpenSSH daemon illustrated in section 3.2.6;
- `sigaction()` acquires and releases a mutex around the call to the original function.

Most wrappers are mutually exclusive because they manipulate the same data structures, for instance the thread list. As a result, multi-threaded programs that frequently call these functions may have slightly reduced performance since synchronization serializes control flow.

Panic Thread Overview

```
static void *
panic_handler (void *args)
{
  trigger_wait ();

  int skip_erasure = 0;
  pthread_mutex_lock (&callback_mutex);
  if (callback)
    skip_erasure = callback ();
  pthread_mutex_unlock (&callback_mutex);

  disable_other_threads ();

  if (!skip_erasure)
    erase_memory (mapsfilefd);

  _exit (0);                    /* fini */
  return NULL;                  /* make compiler happy */
}
```

Listing 3.7: Top-level function for the panic thread.

As shown in listing 3.7, the panic thread first starts with waiting for a trigger, i.e. calling `read()` on the socket. The call blocks as `panicd` does not respond to it, and it fails when `panicd` receives a trigger signal and the connection is closed. As a consequence, no additional CPU time is needed during normal (non-triggered) operation.

After the thread is being awoken, a possibly set callback handler is executed, see section 3.2.1.

The third step calls a function to put all threads but `panic_thread` into `sleep()`. This measure prevents severe errors during erasure of the threads' memory, for example race conditions or `SIGILL` errors that may lead to the process being aborted and cryptographic keys remaining exposed in memory. The `disable_other_threads()` function works by assigning a signal handler with `sleep()` to the process and sending each of the other threads a corresponding signal. See section 3.2.3 for an elaborate reasoning on the use of signals.

Finally, either `_exit()` or the built-in memory erasure function is called. In the latter case, the program does not return from the function.

Built-in Memory Erasure

```
void
erase_memory (const int mapsfilefd)
{
  char buffer[BUFFER_SIZE];
  memset (buffer, 0, sizeof (buffer));

  unsigned int num_entries;
  num_entries = count_erasable_regions (buffer, BUFFER_SIZE-1,
                                        mapsfilefd);
  num_entries = 2 * (num_entries + 1);
  struct proc_maps_entry entries[num_entries];
  memset (&entries, 0, num_entries * sizeof (entries[0]));

  proc_maps_read (entries, num_entries,
                  buffer,  BUFFER_SIZE - 1, mapsfilefd);

  /* clear all private pages */
  for (unsigned int i = 0; i < num_entries; ++i)
    if (entries[i].addr_start && entries[i].addr_end
        && PRIVATE == entries[i].type)
      mymemset ((void *) entries[i].addr_start, 0,
                entries[i].addr_end - entries[i].addr_start);

  /* clear all shared pages */
  for (unsigned int i = 0; i < num_entries; ++i)
    if (entries[i].addr_start && entries[i].addr_end
        && SHARED == entries[i].type)
```

```
    mymemset ((void *) entries[i].addr_start, 0,
              entries[i].addr_end - entries[i].addr_start);

  /* force SIGSEGV to cleanup the program
     -- we won't return from here! */
  int *null_ptr_violation = NULL;
  *null_ptr_violation = 42;
}
```

Listing 3.8: Built-in memory erasure (simplified).

The built-in memory erasure function is executed, if either no callback handler is assigned or if it is set and the returned value is `0`.

First of all, variables are allocated on the panic thread's stack to avoid delayed memory allocation which could result in the page mappings being altered.

Second, the number of relevant memory regions is determined during the first read pass through the `maps` file. All writeable regions except the panic thread's own stack are counted. Then, a buffer for more than twice as many memory regions as previously determined is allocated. This preallocation scheme should ensure a large enough buffer when the file is read again.

Third, the `maps` file is read a second time and the final memory region boundaries for erasure are stored in the buffer. After this point it is not possible anymore to call external functions as their correct execution cannot be guaranteed due to access of invalid data.

Fourth, memory regions are overwritten using a local `memset()`-like function. Private pages are erased before shared pages as the latter might cause other processes accessing such a page to fail. However, if all these processes use `libpanic`, the library has more time to process the trigger signal and stop all relevant threads, thus, more likely preventing crashes.

Finally, a null pointer access provokes a segmentation fault and forces removal of the process by the kernel.

3.2.3. Rationale

Implementation of libpanic Functionality as Thread

The main functionality in `libpanic`, for example reading from the socket and erasure of memory, is executed as thread. This essential design decision succeeded compared to two alternatives:

- `ptrace()` with `PTRACE_POKEDATA`, and
- writing to `/proc/[pid]/mem` from another process.

The `ptrace()` syscall is used by debuggers, such as the GNU Debugger[3], to attach to another process and control its execution. Using the `PTRACE_POKEDATA` command [23], it is possible to write data into the process memory. As such, it is suitable for memory erasure; however, a process can detect being traced and change its behaviour or disable debugger attachment [26, Line 1999].

The latter option, involves writing to the `/proc/[pid]/mem` file from another process. As allowing processes to access each other's memory is generally considered a security flaw [14], it is usually not possible and prohibited by the kernel especially on hardened systems.

Therefore, the only reasonable option is to run `libpanic` within the process to be protected, i.e. as thread.

Disabling of other Threads

As `libpanic` creates its own thread, each program it is loaded into is multi-threaded with at least two threads. If memory were erased while other threads are still running, the complete process including panic thread could crash because of undefined behaviour resulting from threads accessing erased memory. As a result of such a crash (which would terminate the panic thread early), part of the original content could remain in memory.

Alternatively, it is possible to kill individual threads using `pthread_cancel()`. Thereby, only the panic thread survives and can access the memory exclusively. Even though this approach seems attractive, not all writeable process memory can be erased with this method: each thread has its own stack and those pages are unmapped when the thread is cancelled; henceforth, a cancelled thread's stack cannot be erased.

As outlined, it is clearly necessary to stop or halt the execution of non-panic threads while they stay in memory. During development, the two following options were considered but needed to be rejected:

- `vfork()`, and
- sending of `SIGSTOP` to specific threads.

[3] `https://www.gnu.org/software/gdb/`

The `vfork()` syscall [25] almost has the same effect as `fork()` [24], i.e. it "creates a child process of the calling process" [25]. In contrast to `fork()`, `vfork()` does not copy the page tables of the parent and suspends it "until the child terminates [...], or [...] makes a call to `execve(2)`" [25]. In principle, `vfork()` therefore could be used to stop non-panic threads by not terminating and not calling `execve()`; nonetheless, "the programmer cannot rely on the parent remaining blocked until the child either terminates or calls `execve(2)`, and cannot rely on any specific behaviour with respect to shared memory" [25]. Moreover, `vfork()` is implemented using copy-on-write pages [25], so even if it had blocked the parent process reliably, memory erasure from the child would not have had overwritten the parent's data, but rather a freshly created copy.

The second option calls `pthread_kill()` to send a `SIGSTOP` signal to all non-panic threads. The advantage using `SIGSTOP` is that it cannot be caught by the receiving process, so the process is stopped reliably. However, even if `SIGSTOP` is sent to a specific thread, it still always stops the whole process including the panic thread.

Finally, the implemented approach is derived from the second option but instead of `SIGSTOP`, `SIGSEGV` is used and an appropriate process wide signal handler is installed. To prevent other threads from overriding the signal handler, a wrapper for `sigaction()` is provided and blocks when the corresponding mutex lock is acquired, see section 3.2.2.

3.2.4. Verification

Simple verification of `libpanic` is conducted using the GNU Debugger and the Beaglebone Black connected to a circuit as depicted in figure 3.3. The pull-up resistor R1 and switch SW1 provide a high or low signal depending on the switch state. In case the GPIO pin on the BBB is misconfigured, for instance as output, resistor R2 prevents excessive currents. The trigger signal is attached to `GPIO0_7` which corresponds to pin 42 on the BBB's P9 expansion header [6, Table 12].

The program to be used for testing is shown in listing 3.9. It creates three threads and prints messages from each thread and the main thread in 1 s periods.

In addition, listing 3.10 illustrates the test procedure in GDB. First, `panicd` and the test program are started. Then, the GPIO pin is triggered and GDB interrupts for each non-panic thread that receives `SIGSEGV` from the panic thread. Next, the entries for the memory regions to be erased are printed to syslog and the program is interrupted at a breakpoint before erasure. Now, an address from the syslog is chosen and its contents are printed. Finally, memory is erased and the same section is printed again showing overwritten data.

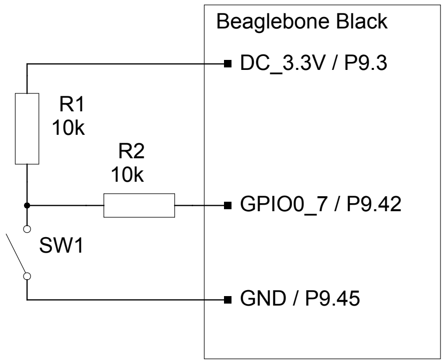


Figure 3.3.: Circuit on a breadboard to simulate the toggle signal.

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/syscall.h>
#include <pthread.h>

#define NUM_THREADS 3

void * printer (void *arg) {
  for (int i = 0; i < 30; ++i) {
    printf ("(%d, %ld) is still running\n", getpid (),
            syscall (SYS_gettid));
    sleep (1);
  }
  return NULL;
}

int main (int argc, char *argv[]) {
  pthread_t th[NUM_THREADS];
  for (int i = 0; i < NUM_THREADS; ++i)
    pthread_create (th + i, NULL, printer, NULL);
  printer (NULL);
```

```
  for (int i = 0; i < NUM_THREADS; ++i)
    pthread_join (th[i], NULL);
  exit (EXIT_SUCCESS);
}
```

Listing 3.9: A simple multi-threaded test program.

```
$ gcc -O2 -g --std=c99 -Wall -Werror -pthread -o some_process
    ↪ some_process.c
$ sudo panicd --gpio 7 --daemon
$ cat gdb.conf
set environment LIBPANIC_DEBUG 1
set environment LD_PRELOAD libpanic.so
set breakpoint pending on
directory ~/libpanic-0.0/src/
break proc_maps.c:282
run
$ gdb -x gdb.conf ./some_process
...
[New Thread 0xb6ec5470 (LWP 686)]
[New Thread 0xb66c5470 (LWP 687)]
(682, 687) is still running
[New Thread 0xb5ec5470 (LWP 688)]
(682, 688) is still running
[New Thread 0xb56c5470 (LWP 689)]
(682, 682) is still running
(682, 689) is still running
...
# push-button is pressed
Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0xb56c5470 (LWP 689)]
...
(gdb) continue
Continuing.

# repeats three more times, once for each thread and the main
    ↪ thread
```

```
(gdb) continue
Continuing.
[Switching to Thread 0xb6ec5470 (LWP 686)]

Breakpoint 1, erase_memory (mapsfilefd=<optimized out>) at
   ↪ proc_maps.c:282
282   for (unsigned int i = 0; i < num_entries; ++i)

# 'tail /var/log/syslog' shows the mapped regions; one line is
   ↪ chosen,
# for example '... b6fce000-b6fcf000 ...
   ↪ /lib/.../libpthread-2.13.so'
(gdb) x/32xw 0xb6fce000
0xb6fce000: 0x00017ef0  0xb6ffa000  0xb6fef984  0xb6fb9b24
0xb6fce010: 0xb6fb9b24  0xb6fb9b24  0xb6fb9b24  0xb6f43ced
0xb6fce020: 0xb6fb9b24  0xb6fb9b24  0xb6fb9b24  0xb6fb9b24
0xb6fce030: 0xb6fb9b24  0xb6fb9b24  0xb6ef5e10  0xb6f61ba0
0xb6fce040: 0xb6fb9b24  0xb6fb9b24  0xb6fb9b24  0xb6f2bcb0
0xb6fce050: 0xb6fb9b24  0xb6fb9b24  0xb6fb9b24  0xb6fb9b24
0xb6fce060: 0xb6f6d4e9  0xb6fb9b24  0xb6fb9b24  0xb6f5e839
0xb6fce070: 0xb6f5f681  0xb6fb9b24  0xb6fb9b24  0xb6fb9b24
(gdb) continue
Continuing.
Cannot find user-level thread for LWP 686: generic error
(gdb) x/32xw 0xb6fce000
0xb6fce000: 0x00000000  0x00000000  0x00000000  0x00000000
0xb6fce010: 0x00000000  0x00000000  0x00000000  0x00000000
0xb6fce020: 0x00000000  0x00000000  0x00000000  0x00000000
0xb6fce030: 0x00000000  0x00000000  0x00000000  0x00000000
0xb6fce040: 0x00000000  0x00000000  0x00000000  0x00000000
0xb6fce050: 0x00000000  0x00000000  0x00000000  0x00000000
0xb6fce060: 0x00000000  0x00000000  0x00000000  0x00000000
0xb6fce070: 0x00000000  0x00000000  0x00000000  0x00000000
(gdb) kill
Kill the program being debugged? (y or n) y
```

```
(gdb) quit
```

Listing 3.10: Verification of `panicd` and `libpanic` using GDB.

3.2.5. Limitations

Naturally, the implementation of `libpanic` imposes some constraints with regard to the application:

1. The library only works with multi-threaded programs that use POSIX threads[4]; a program must not call `clone()` directly;
2. Every program `libpanic` is loaded into is multi-threaded since the library's main functionality is executed in its own thread;
3. A system using the library must not use paging or a swap partition, respectively. As the library does not force all pages into RAM, allowing paging can slow down memory erasure and leaves vulnerable data on the hard disk or flash IC [30], even if overwritten by `libpanic`;
4. The system has to ensure that all writeable private pages physically belong to one process only. If writeable memory, for instance memory of a shared object, is marked as private in the `/proc/self/maps` file of different processes but physically maps to the same shared frame, processes might crash due to invalid data.

3.2.6. Example: OpenSSH Daemon

For most programs, it should suffice to be started using the `LD_PRELOAD` environment variable in order to load `libpanic`. However, some programs change or drop environment variables at some point during execution, for example the OpenSSH daemon[5] `sshd`. In those cases, a simple approach to load `libpanic` is by editing the Makefile and explicitly linking to the library as listing 3.11 illustrates.

```
$ tar xzf openssh-6.7p1.tar.gz
$ cd openssh-6.7p1/
$ ./configure
$ grep -A1 -n -E '^sshd' Makefile
162:sshd$(EXEEXT): libssh.a     $(LIBCOMPAT) $(SSHDOBJS)
```

[4] `man 7 pthreads`
[5] `http://www.openssh.com/index.html`

```
163-    $(LD) -o $@ $(SSHDOBJS) $(LDFLAGS) -lssh
   ↪ -lopenbsd-compat $(SSHDLIBS) $(LIBS) $(GSSLIBS) $(K5LIBS)
# insert `-lpanic' into line 163
$ grep -A1 -n -E '^sshd' Makefile
162:sshd$(EXEEXT): libssh.a     $(LIBCOMPAT) $(SSHDOBJS)
163-    $(LD) -o $@ $(SSHDOBJS) $(LDFLAGS) -lssh
   ↪ -lopenbsd-compat -lpanic $(SSHDLIBS) $(LIBS) $(GSSLIBS)
   ↪ $(K5LIBS)
$ make
$ ldd sshd | grep panic
libpanic.so.0 => /usr/lib/libpanic.so.0 (0xb6f1c000)
```

Listing 3.11: Editing the Makefile of `sshd`.

If this modified version is run as in listing 3.12, it can be seen that each `sshd` process has a panic thread attached. Moreover, the panic thread is preserved across `fork()` boundaries within `sshd`, but it is not added to `bash` and below since these programs neither explicitely link to `libpanic` nor is the `LD_PRELOAD` variable set.

```
$ sudo panicd --gpio 7 --daemon
$ pstree
systemd-+-login---bash---pstree
        |-panicd---panicd
        `-...
$ sudo $PWD/sshd
$ ssh localhost
$ pstree
systemd-+-login---bash---ssh
        |-panicd---panicd
        |-sshd-+-sshd-+-sshd-+-bash---pstree
        |      |      |      `-{sshd}
        |      |      `-{sshd}
        |      `-{sshd}
        `-...
```

Listing 3.12: Propagation of panic thread in `sshd`.

3.3. Memory Erasure Scripts

As previously explained, the purpose of `libpanic` is to erase the memory of selected programs almost immediately after the trigger has been received. In a second phase initiated by the termination of `panicd`, a set of shell scripts is executed and causes erasure of the entire memory.

This feature reuses some scripts from Tails [12], a Debian GNU/Linux distribution with an emphasis on privacy and anonymity. Among other types of media, it can be started from a USB stick and runs as live operating system in RAM. During runtime, a watchdog program checks whether the boot medium is still present. If the user, for instance a journalist, is done with her work, it is sufficient to remove the boot medium and the memory erasure scripts are triggerd, thus, leaving no trace of the activities on the computer.

In both *Panic!* and Tails, an init-premount script is added to the initramfs. The script checks the kernel command line for an `sdmem=` argument, and if found, executes `sdmem` which erases the memory. On Debian, `sdmem` can be obtained from the secure-delete[6] package.

While it is possible to use `sdmem` as is and erase most of the memory, the contents of the running kernel remain vulnerable. Therefore, a `panicd-kexec` init script similar to `tails-kexec` can be started and calls `kexec` from kexec-tools[7] to load a fresh kernel image to memory. Then, `panicd` is started. If it terminates or the `panicd-kexec` script is stopped on shutdown, `kexec` is called again to execute the previously loaded kernel with the added `sdmem=` argument. This way, the memory contents either consist of a fresh kernel image or are erased by `sdmem`.

3.3.1. Limitations

In order to use the `kexec` userspace tools, the kernel has to have been compiled with the `kexec()` syscall enabled. Besides, the kexec-tools packages in the current Debian stable and testing are too old and lack important features added in recent releases specifically for the ARM platform. As a result, kexec-tools must be compiled and installed from source. For *Panic!*, kexec-tools-2.0.7 is used.

Since the `panicd-kexec` init script starts a new kernel, memory erasure via `sdmem` is not started immediately. During the tests, the delay was between 5 s and 6 s.

[6] `https://packages.debian.org/wheezy/secure-delete`
[7] `http://horms.net/projects/kexec/`

In addition, `panicd-kexec` has several hardcoded parameters, in particular, the paths for the kernel image, initrd file, and device tree blob.

4. Panic-Sense

The panic-sense component of *Panic!* consists of a PCB with similar dimensions as the Beaglebone Black and is used to measure several physical properties and as backup power supply. An Atmel ATxmega32A4U [8] AVR XMEGA [7] microcontroller continuously reads and evaluates the measurements, to then decide whether to trigger the router's GPIO pin. To provide a protected environment, router and panic-sense PCB are placed together in an air-tight and pressurized containment, see chapter 5.

The herein discussed panic-sense v0.2 is a prototype and not yet ready for use besides testing. Nonetheless, it allows to gain experience and to express the limitations described in section 4.5.

4.1. Required Features

4.1.1. Sensors

The following sensor types are required for the *Panic!* system:

- one or more temperature sensors to detect cold-boot attacks;
- one absolute air pressure sensor to detect violations of the containment integrity;
- one three-degrees-of-freedom (3 DoF) acceleration sensor to detect if an attacker moves the entire containment;
- an indicator of the currently active power source to detect if an attacker removed the external supply.

The main integrity indicator is the different absolute air pressure in the containment compared to the environment since it allows to detect attacks very reliably. If the containment is opened by an attacker, the pressures even out almost immediately. The pressure should be selected to be outside the natural range of weather phenomenons and an additional safety margin; of course the concrete value, i.e. whether overpressure or underpressure is used and at which intensity, depends on the containment and the available tools, see chapter 5 for a broader discussion.

Air pressure as a physical property is chosen since it can be easily adjusted and measured. Other properties, for instance light intensity and magnetic field strength, can be imitated by an attacker with simple means, or are more difficult to distinguish between healthy and compromised state, for example gas sensors where it can take a long time for the fluids to mix.

Furthermore, it is important to measure only physical properties that can be trusted, i.e. the sensors and electrical connections are physically inside the containment and do either measure local properties like temperature and air pressure or global static properties like the force of gravity.

The accelerometer is added as input source because otherwise an attacker could place the containment in a pressure chamber, imitate a similar pressure as inside the containment and open it. One might argue that it would be sufficient to use just the accelerometer and temperature sensors; however, the accelerometer is likely to have a high rate of false positives, for instance people walking nearby or doors being opened/closed may cause vibrations. Consequently, the trigger thresholds need to be selected wider than just the noise of the sensor, but still small enough to raise the effort for an attacker to move the containment.

4.1.2. Backup Power Supply

Another essential feature is a backup power supply for the router and panic-sense circuit. It guarantees a sufficient supply voltage for at least the time needed for memory erasure on the router; otherwise, an attacker could remove the external power supply, thereby disabling all monitoring and protection measures.

4.2. Circuit

This section refers to the v0.2 prototype schematic and layout for panic-sense as shown in appendix A. Note that v0.2 has limitations and bugs that require at least another iteration, see section 4.5.

The circuit schematics and PCB manufacturing files are drawn using the gEDA[1] suite of GPL-licensed electronic design automation (EDA) tools.

[1] `http://geda-project.org/`

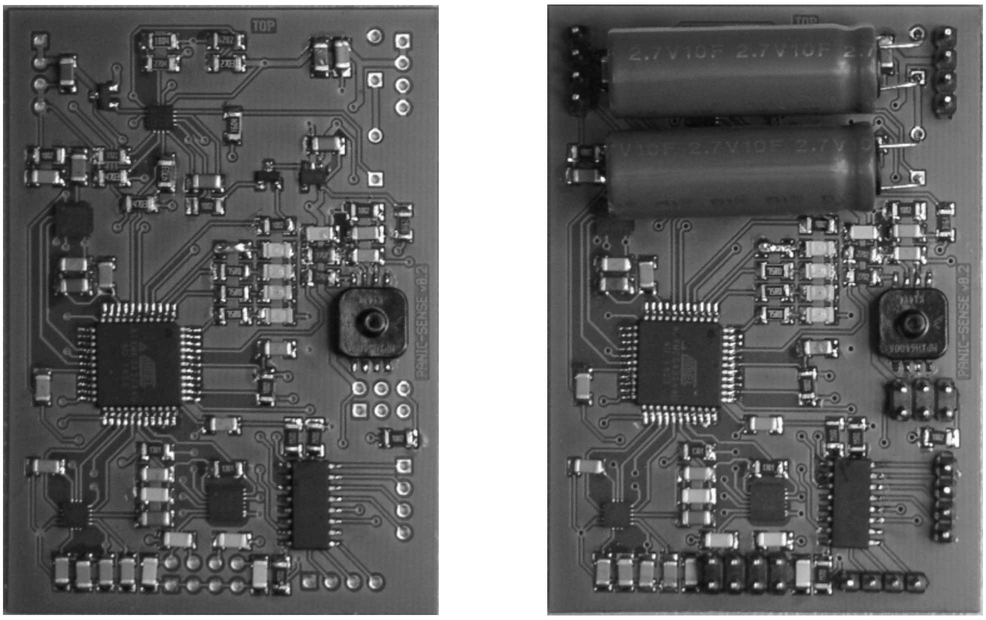


Figure 4.1.: Panic-sense v0.2 PCB with only SMD parts (left), and fully assembled with SMD and THT parts (right).

Figure 4.1 shows the PCBs in different stages of assembly. The two-layer PCBs measure 50 mm by 70 mm with the parts on the top side. Although most parts can be soldered by hand, four parts only exist in small no-lead packages, for instance QFN16, LGA16, and WSON14; thus a reflow oven is needed.

4.2.1. Backup Power Supply

The backup power supply circuit is located on the PCB's upper third and uses two stacked supercaps with 2.7 V/10 F per cell for energy storage. An LTC3226 [11] charges the supercap stack up to 5.3 V after an external supply voltage is attached. Just as stacked lithium-ion polymer battery (LiPo) cells, supercaps need to be balanced during charging, as otherwise the specific surge voltage might be exceeded and is likely to cause permanent damage. Besides actively balancing the supercaps, the LTC3226 provides an ideal diode controller to drive a p-channel MOSFET. Depending on a resistor divider programmable voltage, the external supply voltage is switched on or off via the MOSFET, and a status signal of the active power source is routed to the microcontroller.

In case the external supply is removed, the IC switches over to discharge the supercaps through an internal low-dropout (LDO) regulator. Therefore, an attached router and the panic-sense circuit still receive power for at least 5 s at a current of 1 A, even if no external power source is available. The regulator is capable of supplying a current up to 2 A [11], but as a result of the LDO characteristic it is not possible to deliver a constant voltage of 5 V if the voltage of the supercap stack falls below approximately 5.1 V. This issue is discussed in more detail in section 4.5.

4.2.2. Microcontroller and Sensors

The lower two thirds of the PCB contain the microcontroller and sensors, which are:

- LM95234 [19]: 11 bit temperature sensor with one local channel and four remote channels. For temperature sensing on the remote channels, cheap MMBT3904 [10] transistors are used. They can be soldered on small PCBs and attached to a pin header on the main PCB;
- MPXH6400A [15]: absolute air pressure sensor for 20 kPa to 400 kPa range with analog output;
- FXOS8700CQ [16]: three axis 14 bit accelerometer for ±2 g (smallest measurement range) and three axis 16 bit magnetometer for ±1200 µT;
- L3G4200D [28]: three axis 16 bit gyroscope for $\pm 250\,^\circ\,\mathrm{s}^{-1}$ (smallest measurement range).

Moreover, an optocoupler allows isolated information exchange primarily intended for the trigger signal from the panic-sense circuit to the router, but also for bidirectional UART communication and an additional output reserved for future purposes.

4.3. Microcontroller Software upanic

The `upanic` software is derived from the code[2] of the Fiber-Optic Vibration Sensing Experiment[3] (FOVS), which was launched 2014 as part of RX15[4] mission of the REXUS[5] sounding rocket program.

It is executed by an Atmel ATxmega32A4U [8] AVR XMEGA [7] microcontroller at a CPU frequency of 32 MHz and is responsible for reading and analysing the sensor data, as well as control of the state machine and the output signals.

[2] `https://gitorious.org/fovs/fovsuc`
[3] `http://fovs.de/`
[4] `http://www.dlr.de/rd/en/desktopdefault.aspx/tabid-5282/8854_read-37635/`
[5] `http://www.rexusbexus.net/`

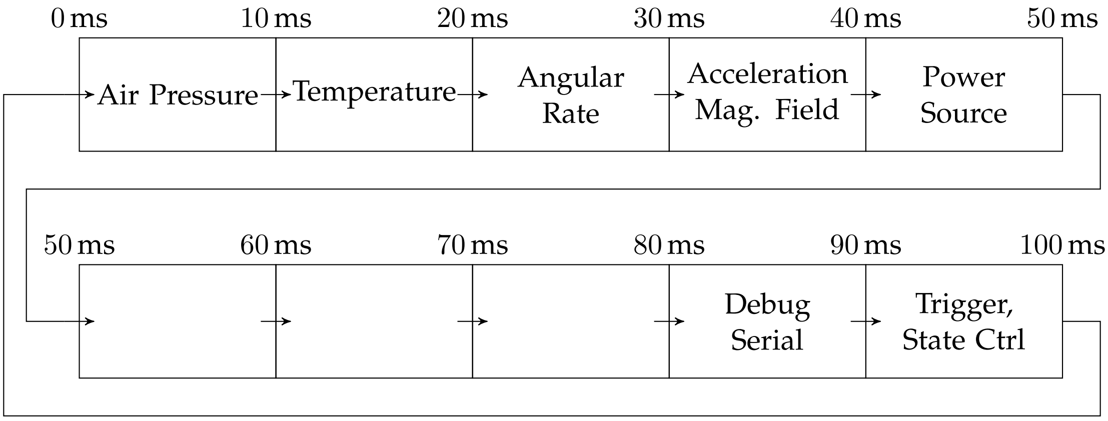


Figure 4.2.: Assignment of tasks to slots.

The current version of the software only supports underpressure environments since it was not possible to find a suited containment for overpressure.

4.3.1. Scheduling

As has been shown by the FOVS experiment [5] for a reliable and easily verifiable approach, a time-triggered static non-preemptive scheduling scheme is used.

A timer periodically generates interrupt requests and causes an interrupt service routine (ISR) to be executed. Inside the ISR, a counter keeps track of the current slot and determines the task to be executed from an array of function pointers. As illustrated by figure 4.2, a slot has a length of 10 ms and a cycle repeats every ten slots or 100 ms, respectively. Currently, the unallocated slots are not needed and reserved for future use.

Unlike event-triggered systems, the static and time-triggered nature of the scheduling simplifies verification of real-time constraints and sensor sampling rate, even in overload situations. Assuming immediate response to an event by the sensors, the worst case latency to a reaction in software is determined by the time for one scheduling cycle (100 ms) and an additional small jitter in the magnitude of a few tens of CPU cycles between interrupt request and call of the ISR.

The scope of the functionality of tasks is strictly isolated: tasks located at the beginning of a cycle initialize the sensors, read their measurements and store the data in the microcontroller's RAM, whereas the last task of a cycle processes these data and decides whether to trigger output signals. Moreover, an additional task is used to simplify testing by emitting the current data through the UART interface.

4.3.2. State Machine

While input oriented tasks provide a generic interface of the respective sensor, the trigger and state control task implements the *Panic!*-specific decision rules and behaviour. After power-on-reset the task executes the following state machine:

1. The status LEDs light up for 2 s and show the user that the system is powered-on. During this time, the sensors are initialized and sampling is started. The trigger line is pulled low to signal the router that the system is not yet in a safe state.

2. The current air pressure is measured for a period of 6.4 s and the average environmental pressure p_{env} is calculated. Using p_{env}, the thresholds are determined by equations 4.1 and 4.2.

$$p_{\text{rising}} = p_{\text{env}} - 200\,\text{hPa} \tag{4.1}$$

$$p_{\text{falling}} = p_{\text{rising}} \cdot 0.75 \tag{4.2}$$

   Since the environmental pressure depends on the elevation of the current location, it is generally not possible to set static limits during compile time. Therefore, the thresholds are determined during runtime using the environmental pressure as reference. Nonetheless, an absolute difference of 200 hPa for the rising limit is enforced.

3. The system is now ready to be evacuated and only switches into the next state if the interior pressure falls below the p_{falling} limit.

4. Once the p_{falling} limit is reached, the system tries to determine the end of evacuation process. If the pressure does not change by more than 50 hPa within a 30 s time frame, the system advances to the next state.

5. To allow the user to remove attached tubes, tools needed during evacuation and to place the system to its final location, it waits for 60 s.

6. The system is at its final location and must not be moved anymore.

7. Similar to the beginning, the accelerometer and magnetometer values are averaged for 6.4 s. The system triggers, if values outside the boundaries of equations 4.3 and 4.4 occur. Likewise, if any measured temperature is out of range

of equation 4.5 or the backup power supply is active.

$$|a_{\mathrm{avg}} - a| \leq 0.1\,\mathrm{g} \tag{4.3}$$

$$|B_{\mathrm{avg}} - B| \leq 50\,\mathrm{\mu T} \tag{4.4}$$

$$5\,^{\circ}\mathrm{C} \leq \vartheta \leq 85\,^{\circ}\mathrm{C} \tag{4.5}$$

By the end of this state, the trigger line is pulled high, indicating that services on the router can be started.

8. The status LEDs start flashing regularly as the system is operational and in safe condition.

9. As the pressure increases over time due to leakage or because of an attack, the trigger line is pulled low as soon as the interior pressure exceeds the p_{rising} threshold.

Note that `upanic` as described here does not permit to re-pressurize the containment as part of maintenance because it cannot distinguish between the device owner attaching tubes and causing vibrations and an adversary. To mitigate this restriction, a modification to the circuit is necessary, see section 4.5.2.

4.4. Cost Estimation

Providing own hardware equipped with sensors and backup power supply, yields expenses for electronic parts, manufacturing of PCBs, and additional tools. If these costs can be kept low, it enables more people to assemble their panic-sense circuit and physically secure their router.

Naturally, cost per piece tends to decline the larger the quantity of purchased parts are. Therefore, cost estimations for a single panic-sense circuit as well as for 50 circuits are listed in tables 4.1 and 4.2. The prices for electronic parts are taken from the Farnell element14[6] distributor in Germany and prices for PCB manufacturing from Multi Circuit Boards[7] at 2014-10-13.

Nonetheless, the information in these tables does not guarantee to yield the lowest price possible and does not include additional costs for tools like soldering equipment.

[6] `http://de.farnell.com/`

[7] `http://www.multi-circuit-boards.eu/`

4.5. Limitations

The panic-sense v0.2 circuit is a prototype with several limitations that prevent its use in a productive environment. Still, the experiences gained while working with this version allow to describe the problems to be solved by the next iteration and the additional capabilities that it needs to provide.

4.5.1. Backup Power Supply

First of all, the current circuit does not provide protection against reverse input voltage, overvoltage, and overcurrent. As a result, it might be possible for an attacker to apply an input voltage that destroys the LTC3226 backup supply controller, but not the voltage regulators on the Beaglebone Black.

Second, there is currently no mechanism to electrically switch on or off an attached router. While this is safe, the ability to control the power state of the main load can be useful, for instance to enable the router after initial pressurization of the containment.

Third, the voltage in backup supply mode is regulated by a low-dropout (LDO) regulator; hence, it cannot provide voltages above the input voltage from the supercaps and the output voltage decreases as the supercaps are being discharged. The Beaglebone Black is still operational below an input voltage of approximately $4.2\,\mathrm{V}$; however, other routers may be more sensitive and may brown-out too early for the memory erasure to complete. Depending on the type of backup energy storage, for example batteries or supercaps, a successor design could use a buck-boost switching regulator to guarantee the correct operational voltage for the router even when no external supply is attached.

Fourth, the air pressure sensor uses the same $5\,\mathrm{V}$ voltage rail than the router, which causes voltage ripple in the supply voltage to show in the measurements. As a consequence, a successor circuit shall provide a separate voltage regulator for the air pressure sensor. In particular, charge pumps seem to be suited.

Last, the current circuit is designed for a router supply voltage of $5\,\mathrm{V}$ only because it is intended as proof-of-concept using a Beaglebone Black. In the future, the panic-sense circuit should be compatible with ordinary home routers that run at $12\,\mathrm{V}$ supply voltage and have higher power requirements.

4.5.2. Microcontroller and Sensors

An important feature to add is an isolated input from the router to the panic-sense circuit. This line can be used to instruct `upanic` to temporarily inhibit triggering although sensors exceed their thresholds, for example, if the containment needs to be re-pressurized and is moved when tubes are attached.

Second, several sensors should be replaced. For instance, the gyroscope is one of the most expensive parts in the circuit, yet it does not provide much additional information not covered by the accelerometer.

Third, the FXOS8700CQ accelerometer and magnetometer cannot tri-state the MISO pin [16, Page 20] of the SPI bus, thus, only one slave (the sensor itself) can be attached to the bus. Choosing another similar device that supports multiple slaves on the same bus increases extensibility for additional sensors if needed.

Fourth, the LM95234 temperature sensor uses the I^2C bus which requires nodes to exchange ACK and NACK bits and allows them to pause the communication (clock stretching). While I^2C may be suited for a variety of applications, it complicates verification of hard real-time deadlines and the implementation of the respective `upanic` task because of its interactive nature and the resulting number of control flows. Instead, it may be simpler to use sensors on the SPI bus, or measure analog voltages via the microcontroller's ADC.

Finally, a separate Ethernet isolator should be added since there is no guarantee that the Ethernet transceivers on a generic router are isolated.

4.5.3. Printed Circuit Board

The current circuit uses four parts in small packages that are difficult to solder and require special tools like a reflow oven. Consequently, these parts should be replaced with parts in packages that can be soldered by hand. In contrast, the packages of most capacitors and resistors could be reduced from size 1206 to 0805 since the latter should still be easy to solder and occupies less PCB area.

In addition, the panic-sense PCB could be split into physically separate modules for the backup power supply and the sensor circuit. The identical sensor circuit could then be reused for different supply circuits, for instance for 5 V and 12 V routers.

Part Name	Ref	Qty	Farnell No	EUR/pcs	EUR	MOQ
ATXMEGA32A4U-AU	U1	1	206-6309	2.8000	2.8000	
L3G4200D	U6	1	187-2924	8.8800	8.8800	
KP-3216SURCK	D1	1	229-0335	0.0920	0.0920	
KP-3216CGCK	D3	1	229-0333	0.1150	0.1150	
KP-3216SYCK	D2	1	229-0336	0.1120	0.1120	
KP-3216QBC-D	D4	1	221-7976	0.1870	0.1870	
FXOS8700CQR1	U5	1	237-7757	3.0700	3.0700	
HV1030-2R7106-R	C10, C11	2	214-8486	4.4400	8.8800	
SI2333CDS-T1-GE3	Q1	1	177-9259	0.4410	0.4410	
LTC3226EUD#PBF	U3	1	203-3980	5.2800	5.2800	
XC6222D331MR-G	U4	1	183-0952	0.4590	0.4590	
MMBT3904	Q2-Q5	4	984-6727	0.0581	0.2324	
ACPL-247-500E	U2	1	163-4758	1.4000	1.4000	
MPXH6400AC6T1	U7	1	223-8141	8.8600	8.8600	
LM95234CISD	U8	1	155-4779	1.6500	1.6500	
BAS40-05,215	D5	1	873-4313	0.0700	0.3500	*
WCR1206-10KFI	R1,R9,R19-R23	6	110-0218	0.0590	0.5900	*
MC0125W120612M70	R13	1	214-2359	0.0530	0.0530	
MC0125W120611M60	R11	1	214-2348	0.0530	0.0530	
CR1206-FX-1004ELF	R14	1	233-3552	0.0530	0.0530	
CRCW120662K0FKEA	R16	1	165-3159	0.0320	0.0320	
WCR1206-150RFI	R6-R8	3	110-0169	0.0140	0.1400	*
CR1206-FX-75R0ELF	R2-R4	3	233-3550	0.0510	0.1530	
MC0125W1206133K2	R10	1	214-2264	0.0530	0.0530	
MC0125W120613K16	R22	1	214-2208	0.0530	0.0530	
WCR1206-27KFI	R24-R25	2	110-0229	0.0150	0.1500	*
WCR1206-1K3FI	R26-R27	2	110-0195	0.0140	0.1400	*
C1206C476M8PACTU	C6	1	157-2639	1.1900	5.9500	*
1206YD475KAT2A	C13	1	132-7729	0.8370	4.1850	*
CC1206JRNPOABN470	C22	1	128-4140	0.2340	2.3400	*
MC1206B103K500CT	C17	1	175-9350	0.0380	0.3800	*
MC1206F474Z250CT	C18	1	175-9321	0.0430	0.4300	*
MC1206B106K160CT	C5,C7-C9,C20,C25	6	232-0921	0.1290	1.2900	*
MC1206N101J500CT	C26-C30	5	175-9327	0.0470	0.4700	*
MCPWR06FTEO4703	R12,R17-R18	3	188-7522	0.0210	0.5250	*
12065C104MAT2A	C1-C4, C12, C14-C16, C19,C21,C23-C24	12	233-2881	0.1990	2.3880	
MCPWR06FTEO2703	R15	1	188-7514	0.0220	0.5500	*
Sum Parts (excl. VAT)					**62.7864**	
Wires, Pin Headers,. . . (est.)					16.2136	
PCB Manufacturing		1			41.0000	
Sum (excl. VAT)					**120.0000**	
Sum for a single unit (excl. VAT)				120.00		

Table 4.1.: Bill of materials and estimation of part and manufacturing cost for a single panic-sense v0.2 PCB.

Part Name	Ref	Qty	Farnell No	EUR/pcs	EUR	MOQ
ATXMEGA32A4U-AU	U1	50	206-6309	1.8500	92.5000	
L3G4200D	U6	50	187-2924	6.8900	344.5000	
KP-3216SURCK	D1	50	229-0335	0.0770	3.8500	
KP-3216CGCK	D3	50	229-0333	0.1010	5.0500	
KP-3216SYCK	D2	50	229-0336	0.0862	4.3100	
KP-3216QBC-D	D4	50	221-7976	0.1460	7.3000	
FXOS8700CQR1	U5	50	237-7757	2.6400	132.0000	
HV1030-2R7106-R	C10, C11	100	214-8486	3.6200	362.0000	
SI2333CDS-T1-GE3	Q1	50	177-9259	0.3660	18.3000	
LTC3226EUD#PBF	U3	50	203-3980	3.5000	175.0000	
XC6222D331MR-G	U4	50	183-0952	0.3780	18.9000	
MMBT3904	Q2-Q5	200	984-6727	0.0270	5.4000	
ACPL-247-500E	U2	50	163-4758	1.2300	61.5000	
MPXH6400AC6T1	U7	50	223-8141	7.8200	391.0000	
LM95234CISD	U8	50	155-4779	1.5700	78.5000	
BAS40-05,215	D5	50	873-4313	0.0590	2.9500	
WCR1206-10KFI	R1,R9,R19-R23	300	110-0218	0.0360	10.8000	
MC0125W120612M70	R13	50	214-2359	0.0240	1.2000	
MC0125W120611M60	R11	50	214-2348	0.0240	1.2000	
CR1206-FX-1004ELF	R14	50	233-3552	0.0390	1.9500	
CRCW120662K0FKEA	R16	50	165-3159	0.0160	0.9000	
WCR1206-150RFI	R6-R8	150	110-0169	0.0090	1.3500	
CR1206-FX-75R0ELF	R2-R4	150	233-3550	0.0230	3.4500	
MC0125W1206133K2	R10	50	214-2264	0.0240	1.2000	
MC0125W120613K16	R22	50	214-2208	0.0240	1.2000	
WCR1206-27KFI	R24-R25	100	110-0229	0.0110	1.1000	
WCR1206-1K3FI	R26-R27	100	110-0195	0.0100	1.0000	
C1206C476M8PACTU	C6	50	157-2639	0.6380	31.9000	
1206YD475KAT2A	C13	50	132-7729	0.4480	22.4000	
CC1206JRNPOABN470	C22	50	128-4140	0.2340	11.7000	
MC1206B103K500CT	C17	50	175-9350	0.0380	1.9000	
MC1206F474Z250CT	C18	50	175-9321	0.0430	2.1500	
MC1206B106K160CT	C5,C7-C9,C20,C25	300	232-0921	0.0880	26.4000	
MC1206N101J500CT	C26-C30	250	175-9327	0.0400	10.0000	
MCPWR06FTEO4703	R12,R17-R18	150	188-7522	0.0180	2.7000	
12065C104MAT2A	C1-C4,C12,C14-C16, C19,C21,C23-C24	600	233-2881	0.0755	45.3000	
MCPWR06FTEO2703	R15	50	188-7514	0.0220	1.1000	
Sum Parts (excl. VAT)					**1883.9600**	
Wires, Pin Headers,... (est.)		50		13.0008	650.0400	
PCB Manufacturing		50		2.8000	140.0000	
Sum (excl. VAT)				**53.4800**	**2674.0000**	
Sum for a single unit (excl. VAT)				53.48		

Table 4.2.: Bill of materials and estimation of part and manufacturing cost for 50 panic-sense v0.2 PCBs.

5. Containment

To physically protect the router and panic-sense circuit from an adversary, both are put in an air-tight containment, which contains joins for a non-return valve and electrical connections. During normal operation, the containment is pressurized, i.e. the internal pressure is either lower or higher than the environment. This difference in air pressure serves as indicator whether the containment is in a safe state or needs to be considered compromised.

5.1. Requirements

Regardless of a concrete realization, each containment should meet some generic requirements:

1. It needs to withstand the pressure it gets exposed to;
2. Its dimensions must be large enough to contain the router and panic-sense PCBs;
3. The leakage rate should be low to reduce overhead for maintenance to regularly pressurize the containment;
4. It should permit creation of joins in the containment surface;
5. It should permit wireless communication to pass;
6. It should be made out of widely available components to simplify reproduction and replacement, for example an object of everyday life or made out of commercial off-the-shelf components;
7. It should be cheap.

5.2. Characteristics of the Environment

The decision between an overpressure or an underpressure environment yields different consequences for the containment as well as the usable hardware.

The use of overpressure permits differential pressures greater than 1 bar; hence, it is easier to distinguish the compromised from the non-compromised state compared to an underpressure atmosphere. There, the upper limit of air pressure is determined by the elevation of the location, and the lower limit by the minimum pressure the containment withstands. Given also the necessity for a safety margin, the range of operational pressure inside the containment is rather small. As a consequence, frequent evacuation of the containment might be necessary, thus, increasing the work load for the user of the *Panic!* system.

Another aspect to consider is reduced heat dissipation in an environment close to vacuum because of reduced convection. In the extreme of no convection, only radiation and heat spreading into the PCB remain. This condition is even stronger than passive cooling, which is met by many home routers.

Also, a bicycle tyre inflator is a widely available tool to create overpressure. Though pumps suitable for evacuation are less common, they can be found in many public institutions, for example water-jet vacuum pumps and diaphragm pumps in physics and chemistry labs at schools and universities.

So in general, an overpressure atmosphere seems to be preferable over underpressure. However, selection of an adequate containment tends to be difficult as the next section explains. In fact, it was not possible during this thesis to identify an appropriate containment for overpressure atmosphere.

5.3. Containment Variants

5.3.1. Aluminium Box

The first type of containment examined is a 220 mm·120 mm·80 mm aluminium box with IP68 ingress protection rating. It consists of a top and a bottom part that can be screwed together and a seal in between, see figure 5.1.

Three holes are drilled in the top part, one for a car valve and two for enamelled copper wires for power supply and panic-sense debug UART. With the wires put in place, the holes are sealed using epoxy adhesive.

An overpressure test was carried out by placing the panic-sense PCB is the box, screwing the parts together, and then pressurizing the containment using a bicycle tyre inflator to an absolute pressure of 6 bar. As this pressure is out of the measurement range of the panic-sense's air pressure sensor, the less accurate manometer of the tyre inflator was used to determine the end pressure. Monitoring the data from panic-sense

Figure 5.1.: Upper part of the aluminium containment with 3 mm thick silicone rubber sponge seal.

showed a rapid decrease in pressure. A further test with the box submerged into water confirmed leakage between the upper and lower part of the box and no leakage at the sealed holes. Within less than 4 h, the pressures evened out.

This test demonstrates that the aluminium box withstands high pressures, but also that the silicone rubber sponge seal is insufficient for an overpressure environment. Due to the high leakage, the box in this configuration is not a suitable containment for *Panic!*. In addition, the metal shields wireless communication and reduces the possible applications of the system.

5.3.2. PET Bottle

PET bottles are very robust and available at practically no cost. These properties make them a commonly used part in water rockets, for instance as pressure vessels for the rocket boosters [20] withstanding pressures above 10 bar. By the same arguments, they are interesting to *Panic!* as well.

While bottles with a large enough cap for the valve and the wires can be found, insertion of the PCBs tends to be difficult. However, there exist techniques and manuals [20] for splicing and reinforcing bottles that afterwards still support high pressures. The author tried to follow these instructions, but did not succeed in building a functioning containment based on PET bottles.

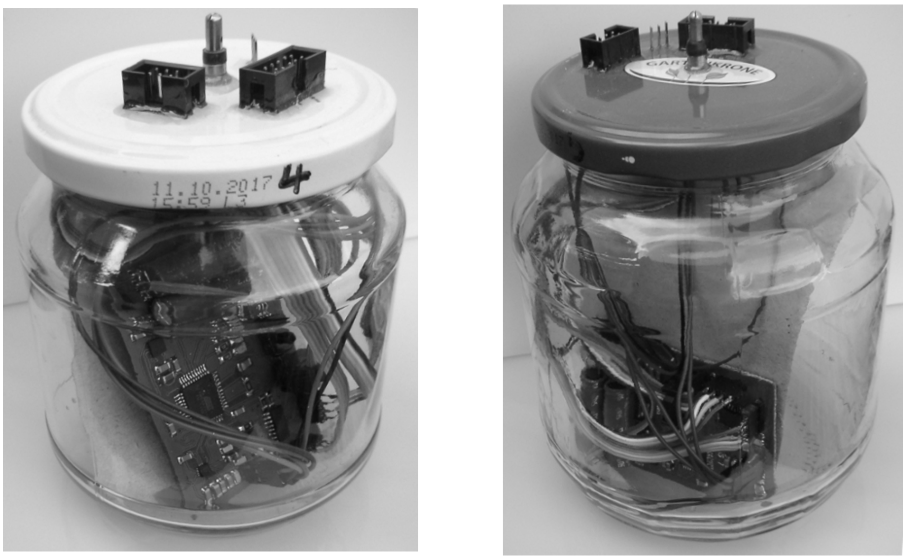


Figure 5.2.: Jars with non-return valve and electrical interface in the cap (left: 580 ml, right: 1 l).

5.3.3. Jar

The last type of containment are jars as commonly used for groceries, for instance jam and gherkins. Jars are built for an underpressure atmosphere and seal themselves when the cap gets sucked in. Moreover, the caps are made out of thin metal that can be modified using a cutter or a hand drill. Figure 5.2 shows some jars altered for *Panic!*.

Since the cap should seal itself, the work focused on the interfaces.

Valve

The first variant depicted in figure 5.3 features a Dunlop bike valve (outward direction of the airflow). Initially, the valve used a short tube that gets expanded when air flows through the valve, and covers the openings when the pressure has the reverse direction. However, the force of the water-jet vacuum pump used for evacuation was too low for the tube to open; hence, the valve was disabled by removing the tube entirely. Instead, a section of a silicone tube with larger diameter was adhered to the cap and blocked after evacuation using a cable strap.

The second variant uses another type of Dunlop valve containing a ball that allows air to pass in one direction and blocks in the other one. As a result, leakage through the valve is lower the greater the differential pressure and vice-versa. If the differential pressure is too low, the valve opens irrevocably.

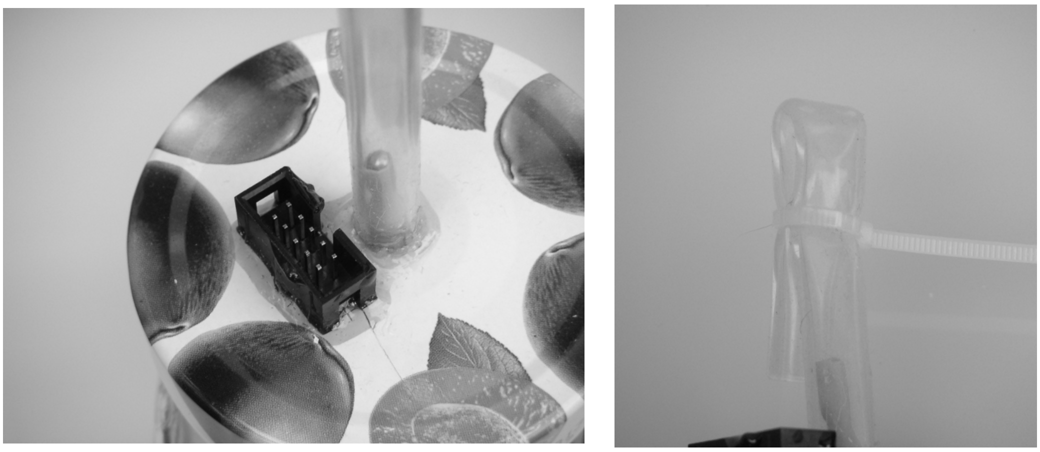


Figure 5.3.: Cap with pin header and blocked tube instead of a valve.

Electrical Interface

The first tests for a containment used enamelled copper wires as electrical interface because they do not have a thick isolation layer where air could leak. That design was dropped for simpler variants since the majority of leakage seems to be caused by cap and valve.

The second variant used stranded wires soldered to a connector adhered to the cap. This option is more user friendly as it uses standard components and reduces the time for preparation of individual enamelled copper wires. Still, it is necessary to drill comparatively large holes for the connector and parts need to be positioned accurately to prevent short-circuit with the cap.

The final variant is a further reduction of previous designs by omitting the connector. Instead, the wires are fed through a slit in the cap as shown in figure 5.4 and sealed using epoxy adhesive. To attach other circuits to the wire, a standard crimp connector can be used.

5.4. Tests

To assess the performance of the jars as containment, several tests were performed by placing a panic-sense PCB for measurement and a power resistor for heat dissipation inside the jar. The tests started with the evacuation of the jar and ended when the pressures evened out. In between, the resistor was activated several times and the measurements were logged.

The plots in figure 5.5 show measurements from the pressure sensor and the PCB

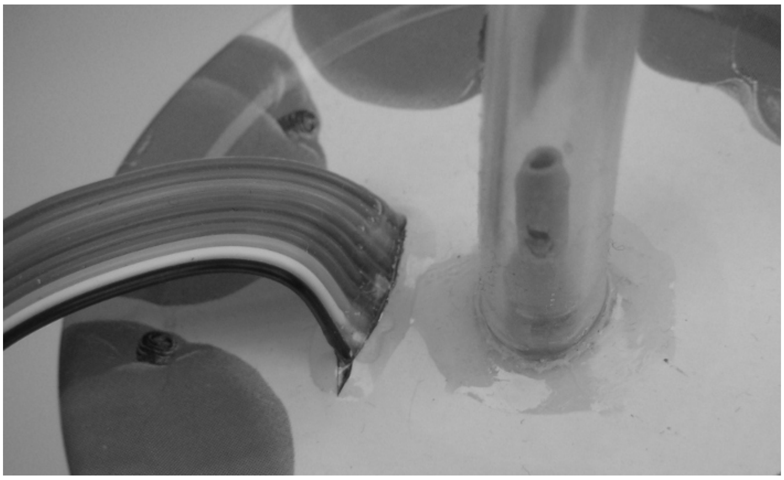


Figure 5.4.: Cap with simplified feedthrough for wires.

temperature sensor. During the first 12 h without thermal test, the pressure increased due to leakage from the containment. When the resistor was enabled and the temperature increased, the pressure increased likewise. The containment tested used the ball based Dunlop valve variant. Therefore, the leakage increased on rising pressure up to the moment of the last thermal test where the minimum differential pressure was exceeded and the valve opened.

In comparison, figure 5.6 depicts the measurements of another test using a jar with tube and cable strap instead of a bike valve. In this configuration, the leakage was much less than in the previous example. Although the start pressure was higher, the test could have been continued for more than two days, if the jar was not opened by removing the cable strap.

Both pressure data sets also show small negative spikes that appear in regular intervals. These are measurement errors caused by the voltage ripple of the sensor's supply voltage. The LTC3226 supercap controller recharges the supercaps to adjust for leakage currents, and thus, causes additional load. As the pressure sensor uses the same voltage rail as the LTC3226, the pressure seems to decrease during this period. This limitation is already described in section 4.5.

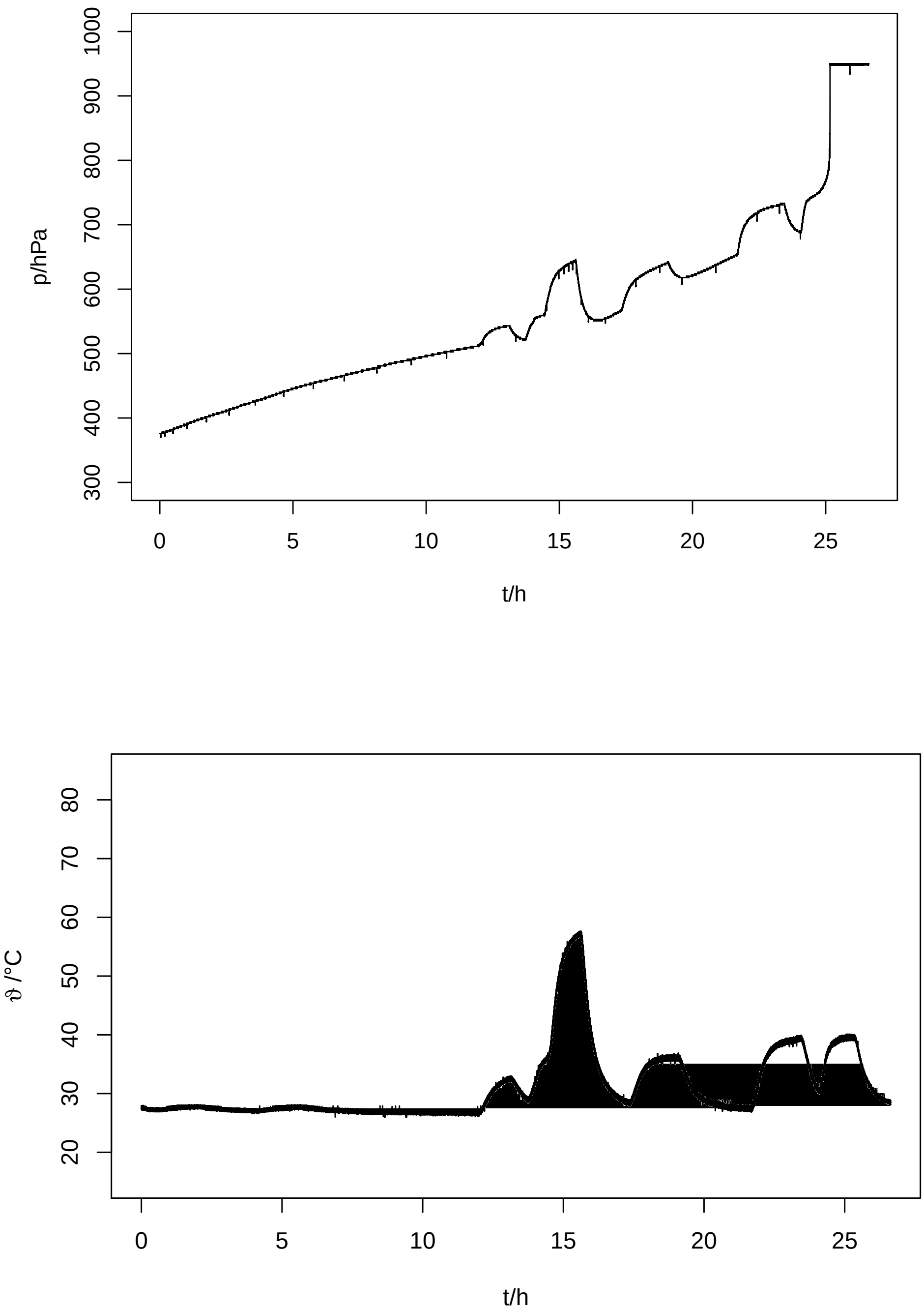
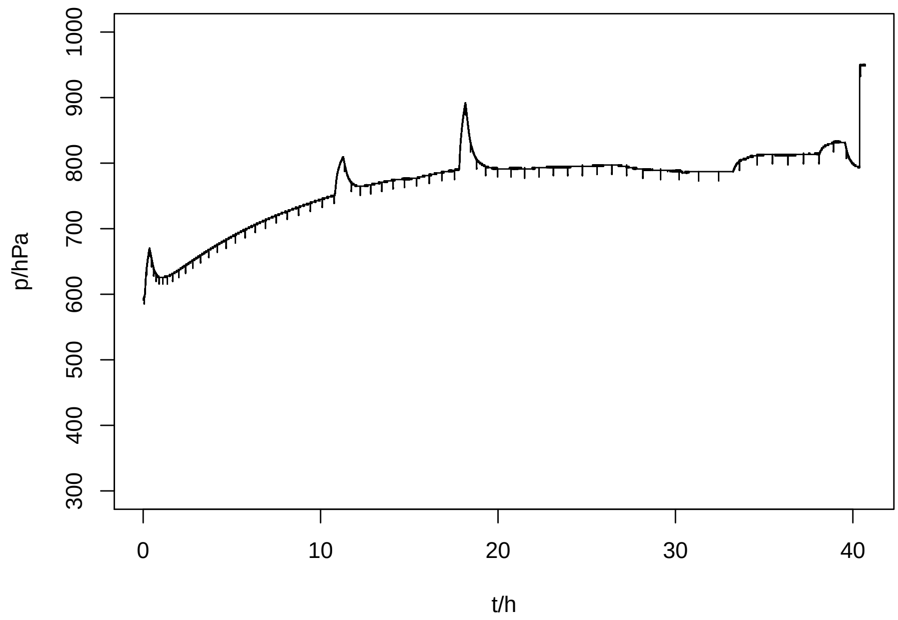


Figure 5.5.: Plots of pressure and PCB temperature measurements (jar size: 580 ml).



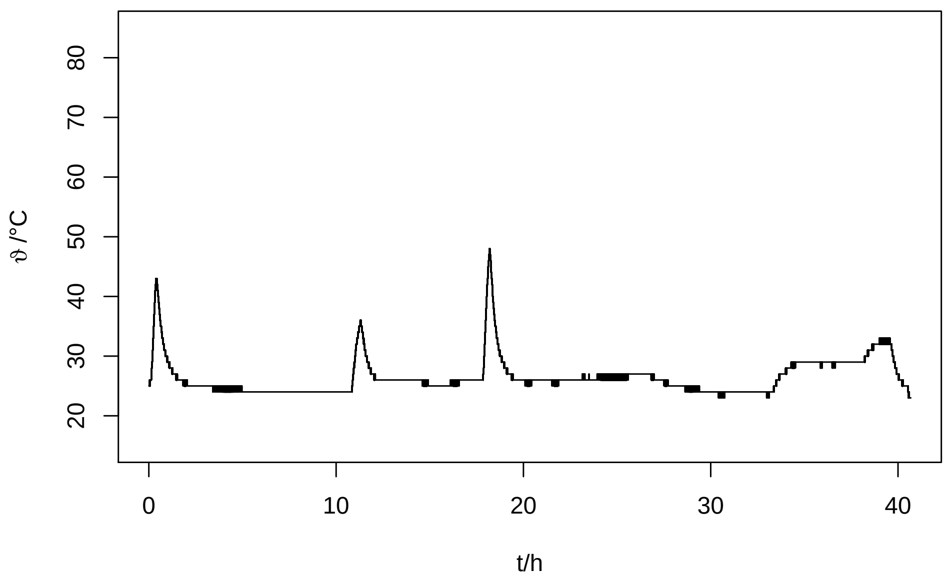


Figure 5.6.: Plots of pressure and PCB temperature measurements (jar size: 400 ml).

6. Conclusion and Future Work

This thesis introduced *Panic!*, a system to physically secure home routers against a wide range of physical attacks. It's components are distributed across various layers of abstraction, ranging from pure software, to software closely coupled with electronics, to pure hardware.

Panic! offers a proof of concept that can be a basis for future work. This includes verification of `panicd` and `libpanic` (especially the memory erasure code), verification of the compatibility between `libpanic` and third-party programs, as well as compatibility tests for non-ARM platforms. Furthermore, another iteration of the panic-sense circuit is necessary to overcome its current limitations and the usability and performance of containments needs to be improved. All three categories gain additional relevance, if *Panic!* shall be made usable with ordinary commercial off-the-shelf home routers and at a larger scale.

A. Appendix

A.1. Panic-Sense Schematics

The following pages show the schematics for panic-sense v0.2.

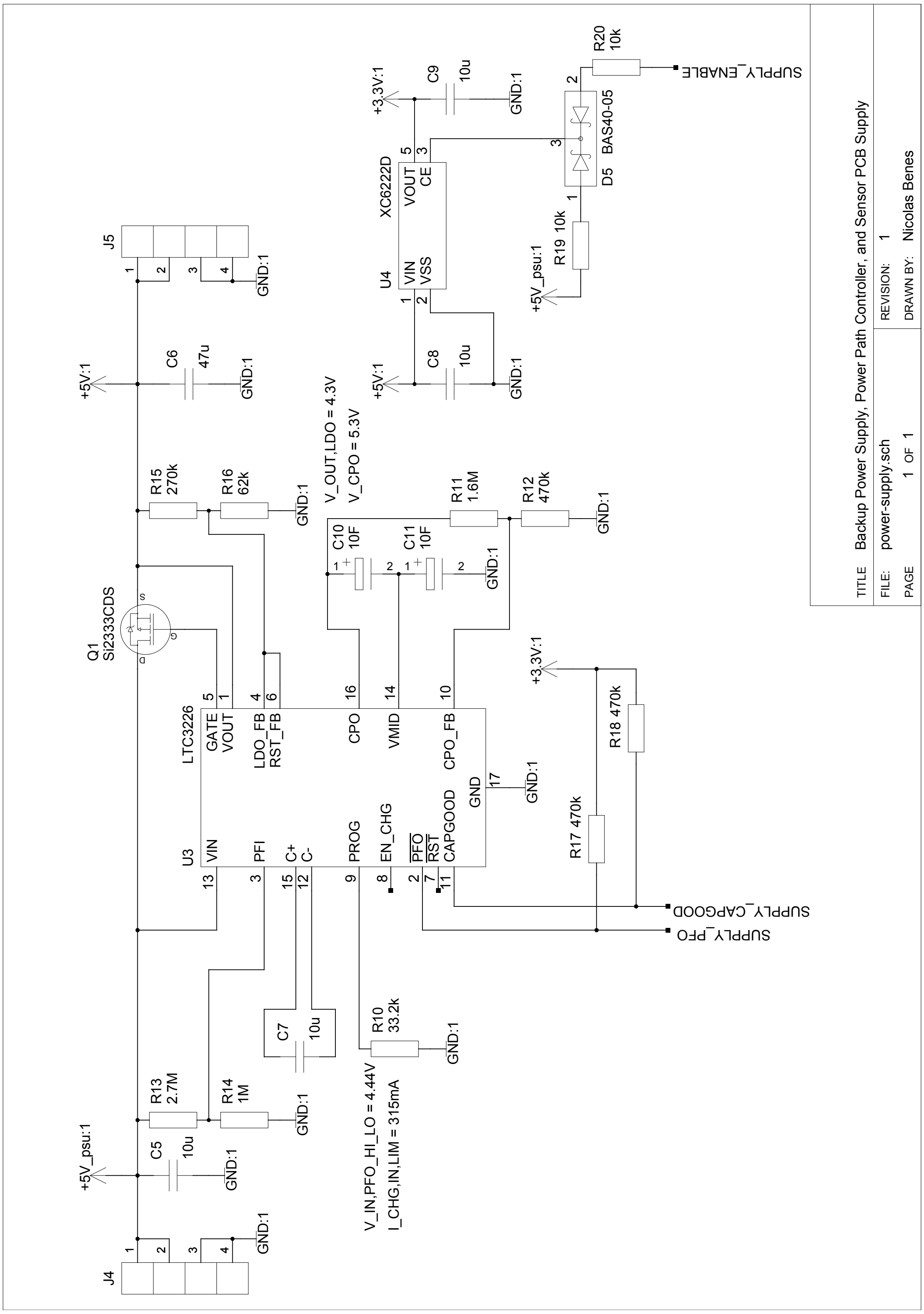


Figure A.1.: Power supply schematic.

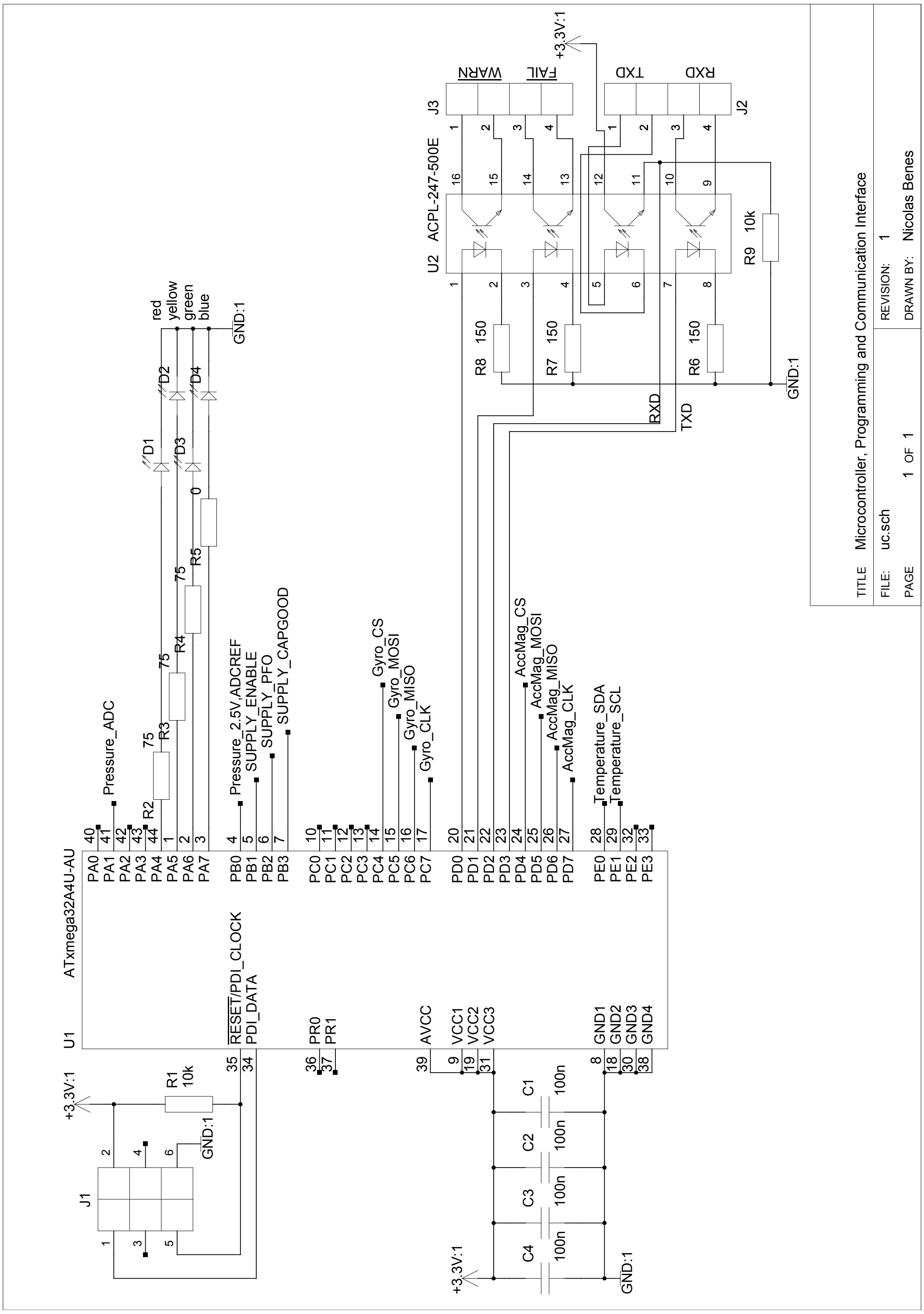


Figure A.2.: Microcontroller schematic.

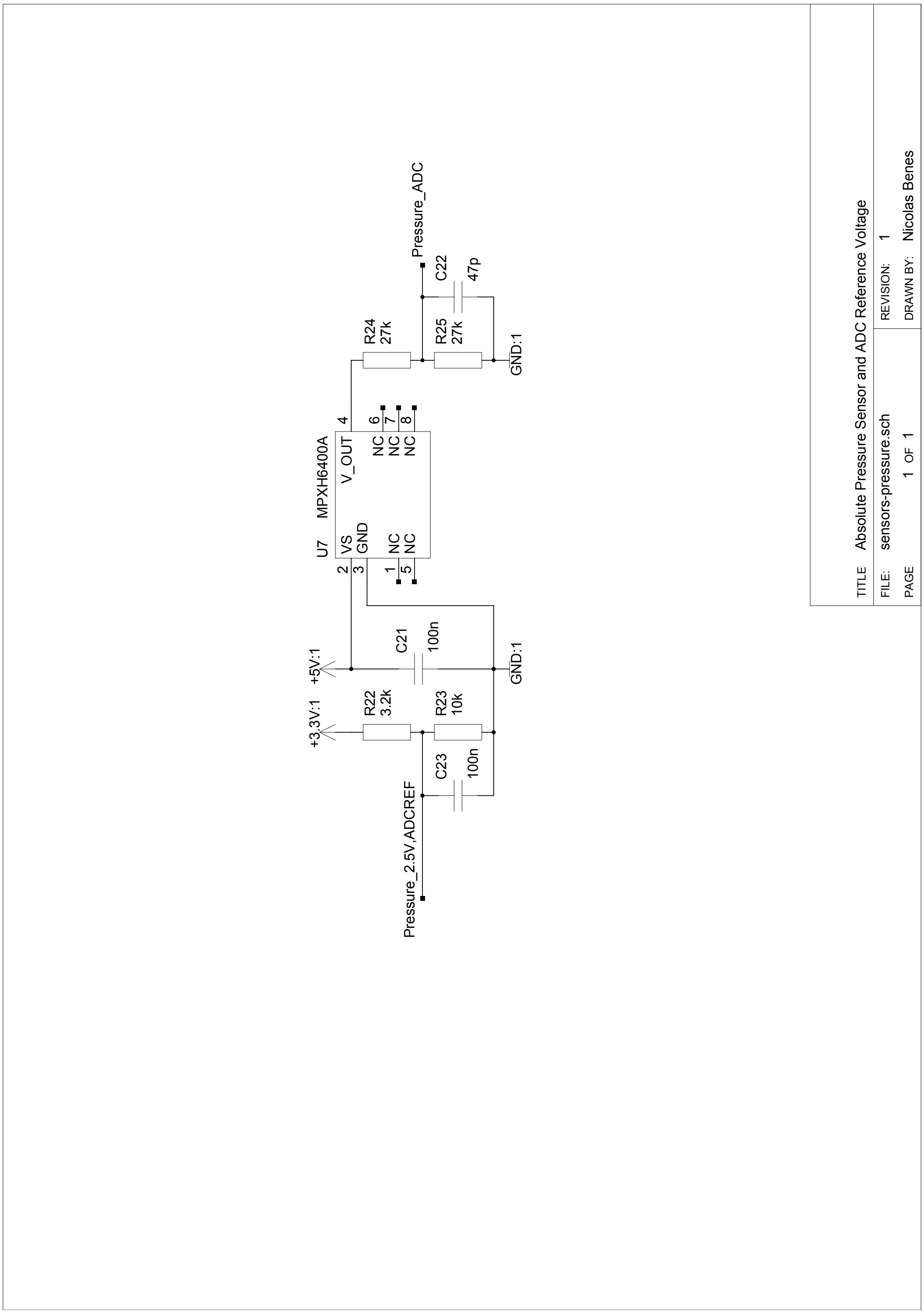


Figure A.3.: Pressure sensor schematic.

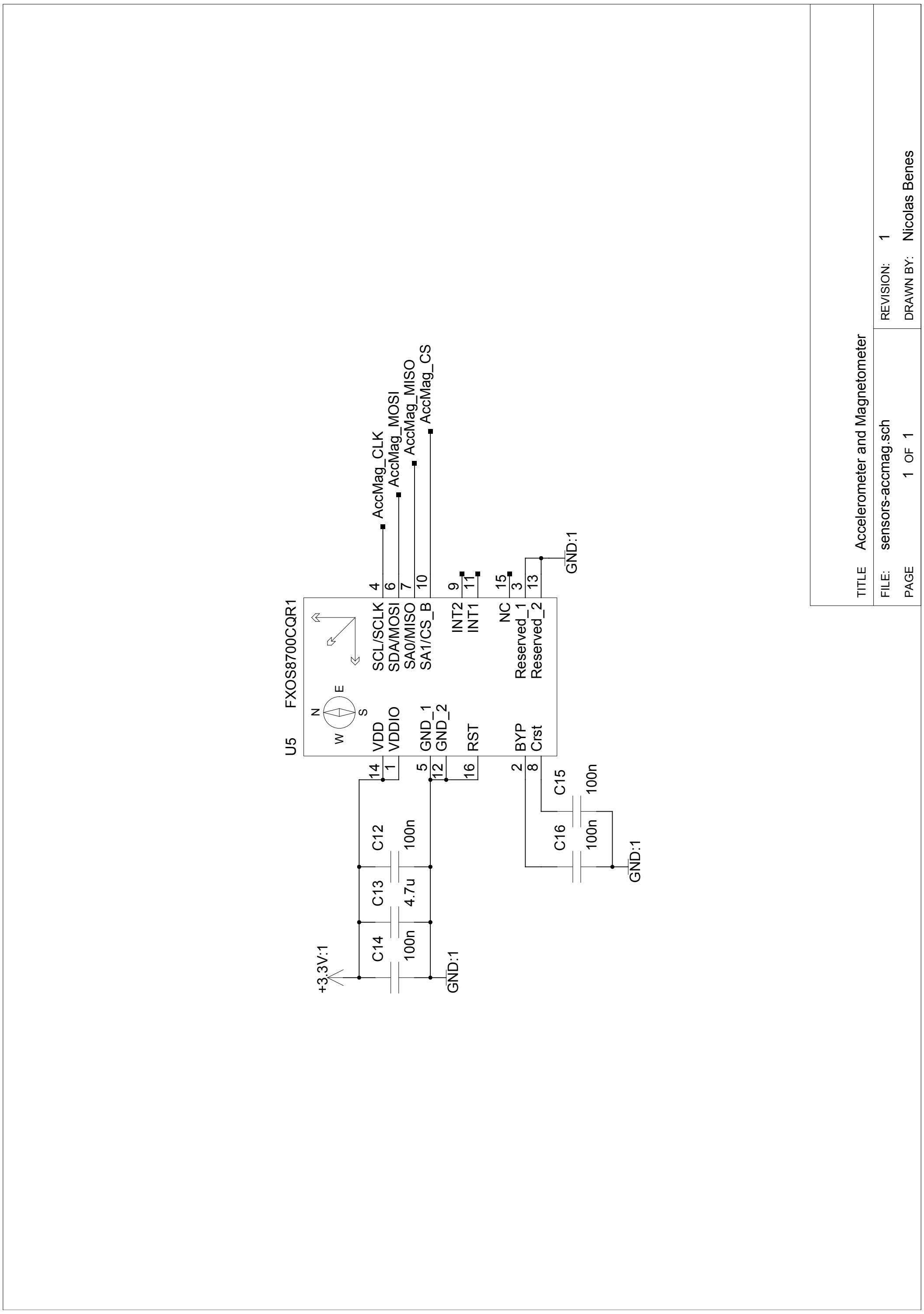


Figure A.4.: Acceleration and magnetic field sensor schematic.

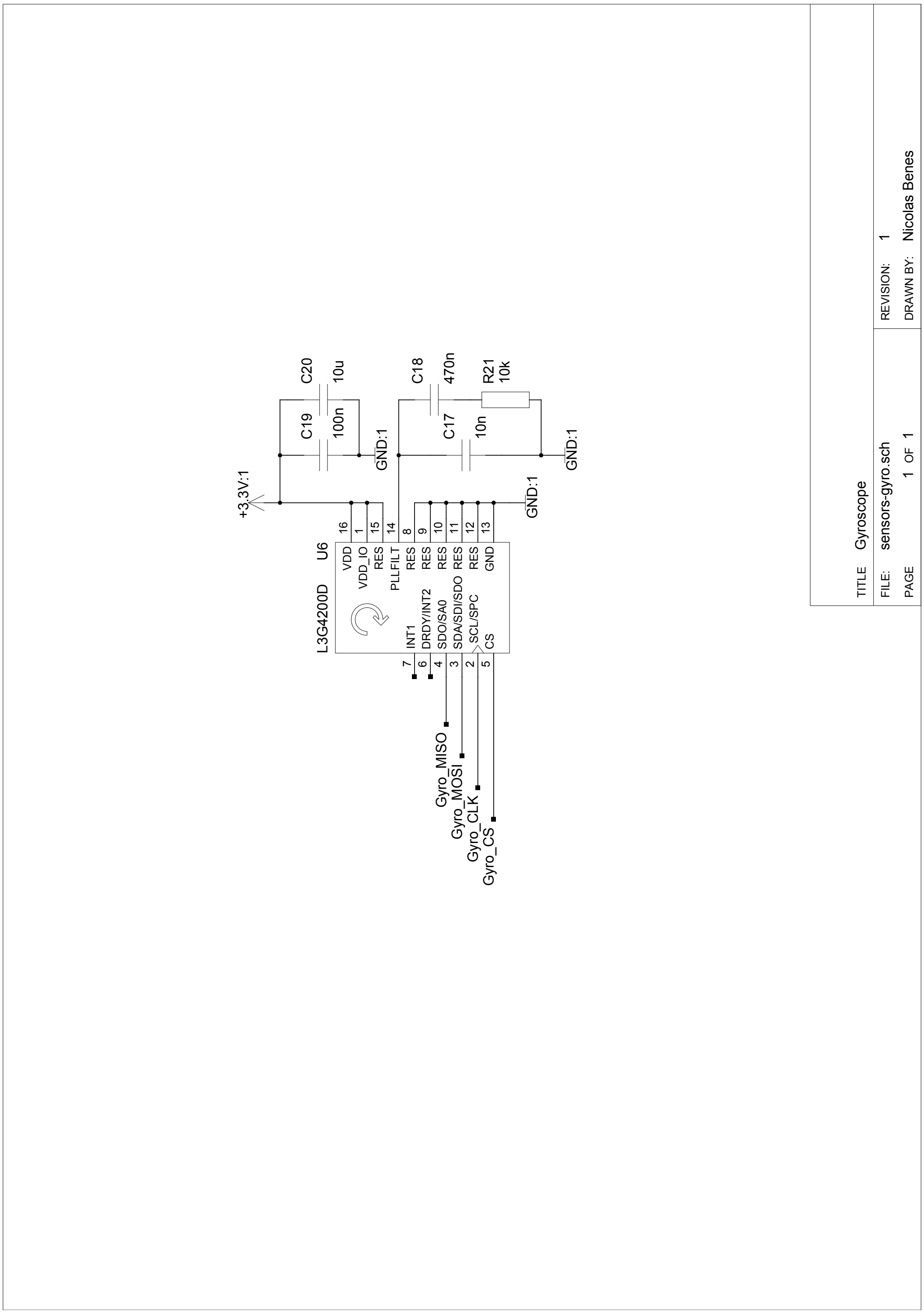


Figure A.5.: Gyroscope schematic.

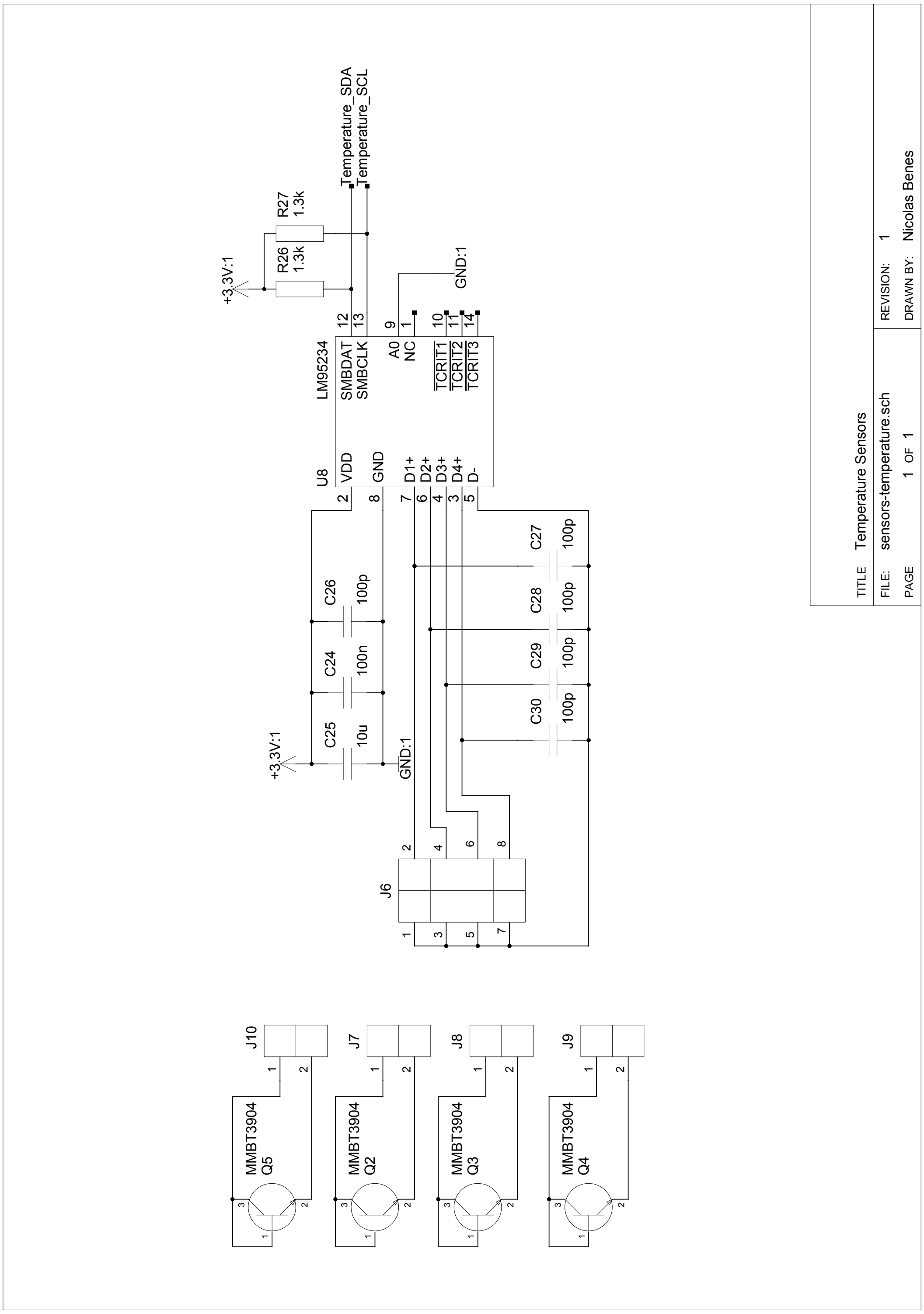


Figure A.6.: Temperature sensors schematic.

B. Bibliography

[1] IEEE Standard for Test Access Port and Boundary-Scan Architecture. *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, pages 1–444, May 2013.

[2] JTAG. `http://wiki.openwrt.org/doc/hardware/port.jtag`, 2014-07-03. Accessed: 2014-08-11.

[3] About OpenWrt. `https://wiki.openwrt.org/about/start`, 2014-07-15. Accessed: 2014-08-10.

[4] Table of Hardware. `http://wiki.openwrt.org/toh/start`, 2014-10-09. Accessed: 2014-10-09.

[5] Nicolas Benes. FOVS – Software. `http://fovs.de/the-experiment/software/`, 2014. Accessed: 2014-10-11.

[6] Gerald Coley and Robert P. J. Day. BeableBone Black System Reference Manual. `https://github.com/CircuitCo/BeagleBone-Black/blob/rev_a5c/BBB_SRM.pdf?raw=true`, 2013-06-15. Accessed: 2014-10-08.

[7] Atmel Corporation. Atmel AVR XMEGA AU MANUAL. `http://www.atmel.com/Images/Atmel-8331-8-and-16-bit-AVR-Microcontroller-XMEGA-AU_Manual.pdf`, April 2013. Accessed: 2014-10-10.

[8] Atmel Corporation. ATxmega16A4U/32A4U/64A4U/128A4U. `http://www.atmel.com/images/Atmel-8387-8-and16-bit-AVR-Microcontroller-XMEGA-A4U_Datasheet.pdf`, October 2014. Accessed: 2014-10-10.

[9] British Broadcasting Corporation. Inside the Dark Web – Episode 4. `http://www.bbc.co.uk/programmes/b04grp09`, 2014. Accessed: 2014-10-09.

[10] Fairchild Semiconductor Corporation. 2N3904 / MMBT3904 / PZT3904 NPN General Purpose Amplifier. `https://www.fairchildsemi.com/datasheets/MM/MMBT3904.pdf`, October 2011. Accessed: 2014-10-10.

[11] Linear Technology Corporation. LTC3226 – 2-Cell Supercapacitor Charger with Backup PowerPath Controller. `http://www.linear.com/docs/40027`, 2011. Accessed: 2014-10-10.

[12] Tails Developers. Tails – Memory erasure. `https://tails.boum.org/contribute/design/memory_erasure/`, 2014-06-26. Accessed: 2014-07-22.

[13] Jeroen Domburg. Hard disk hacking. `http://spritesmods.com/?art=hddhack`, 2013. Accessed: 2014-10-09.

[14] Jason A. Donenfeld. Linux Local Privilege Escalation via SUID /proc/pid/mem Write. `http://blog.zx2c4.com/749`, 2012-01-22. Accessed: 2014-10-07.

[15] Freescale Semiconductor, Inc. High Temperature Accuracy Integrated Silicon Pressure Sensor for Measuring Absolute Pressure, On-Chip Signal Conditioned, Temperature Compensated and Calibrated. `http://cache.freescale.com/files/sensors/doc/data_sheet/MPXH6400A.pdf`, November 2009. Accessed: 2014-10-10.

[16] Freescale Semiconductor, Inc. Xtrinsic FXOS8700CQ 6-Axis Sensor with Integrated Linear Accelerometer and Magnetometer. `http://cache.freescale.com/files/sensors/doc/data_sheet/FXOS8700CQ.pdf`, March 2014. Accessed: 2014-10-10.

[17] Jonathan L. Hafetz. "A Man's Home is His Casle?": Reflections on the Home, the Family, and Privacy During the Late Nineteenth and Early Twentieth Centuries. *Wm. & Mary J. Women & L.*, 8, 2002. `http://scholarship.law.wm.edu/wmjowl/vol8/iss2/2`.

[18] J. A. Halderman, S. D. Schoen, N Heninger, W. Claskson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proc. 2008 USENIX Security Symposium*, 2008-02-21.

[19] Texas Instruments Incorporated. LM95234 Quad Remote Diode and Local Temperature Sensor with SMBus Interface and TruThermTM Technology. `http://www.ti.com/lit/ds/symlink/lm95234.pdf`, March 2013. Accessed: 2014-10-10.

[20] George Katz. Water Rocket Construction - Advanced Tutorials. `http://www.aircommandrockets.com/construction_2.htm`, 2014. Accessed: 2014-10-14.

[21] The Linux man-pages project. *UNIX*, 2008-12-01. `man 7 unix`

[22] The Linux man-pages project. *PROC*, 2009-03-30. `man 5 proc`

[23] The Linux man-pages project. *PTRACE*, 2009-03-30. `man 2 ptrace`

[24] The Linux man-pages project. *FORK*, 2009-04-27. `man 2 vfork`

[25] The Linux man-pages project. *VFORK*, 2009-06-21. `man 2 vfork`

[26] The Tor Project. `common/compat.c`. `https://gitweb.torproject.org/tor.git/blob/HEAD:/src/common/compat.c#l1999`, 2013. Accessed: 2014-10-07.

[27] The Tor Project. Codename: Torouter. `https://trac.torproject.org/projects/tor/wiki/doc/Torouter`, 2014-04-30. Accessed: 2014-08-10.

[28] STMicroelectronics. L3G4200D – MEMS motion sensor: ultra-stable three-axis digital output gyroscope. `http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00265057.pdf`, December 2010. Accessed: 2014-10-10.

[29] Andrew S. Tanenbaum and Todd Austin. *Structured Computer Organization (6th Edition)*. Pearson Education, 2013.

[30] Michael Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. Reliably Erasing Data From Flash-Based Solid State Drives. In *FAST '11: 9th USENIX Conference on File and Storage Technologies*, 2011.