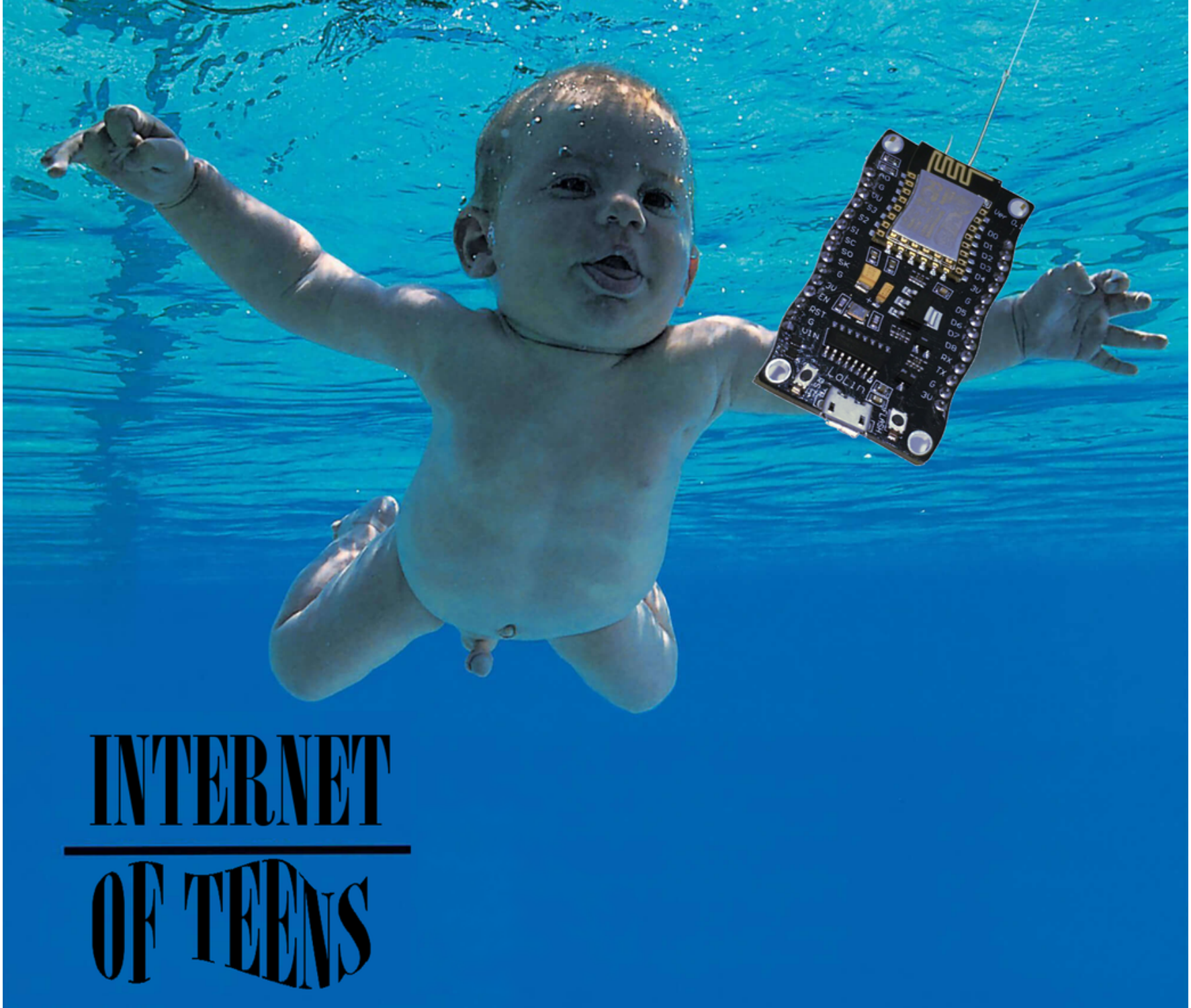

INTERNET
OF TEENS

Smells like Teen Spirit - Internet of Teens @nullcon

Eric Sesterhenn <eric.sesterhenn@x41-dsec.de>

2018

X41 D-SEC GmbH

whoami

- Eric Sesterhenn
- Principal Security Consultant
- Pentesting/Code Auditing at X41
- Father of a daughter

Disclaimer

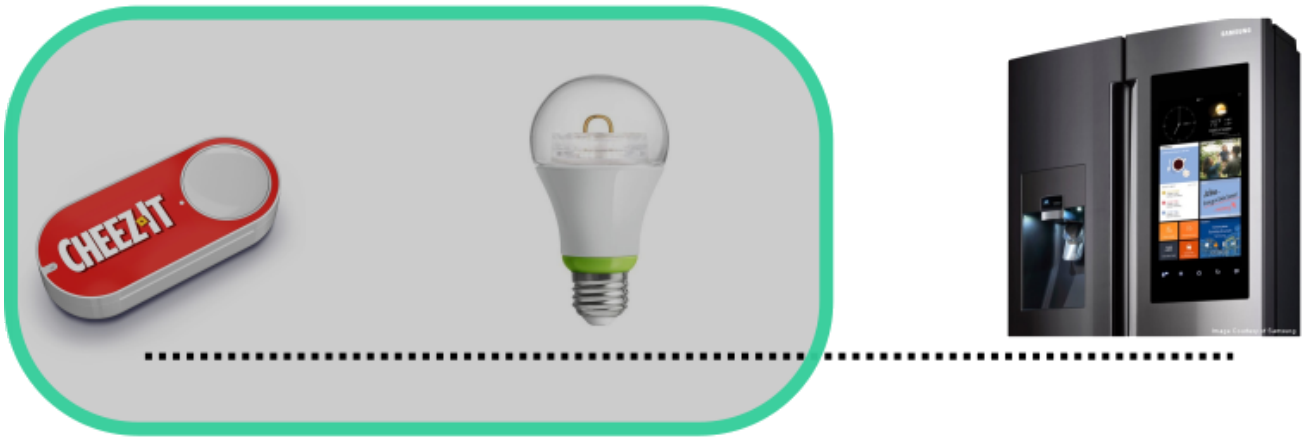
- 0-Day warning!
- None of the issues here has been fixed (checked 14.02.2018)
- Will be reported in the next few days
- Everything found by reading code

IoT and Security

IoT and Security is like teens and sex - everybody talks about it nobody has it

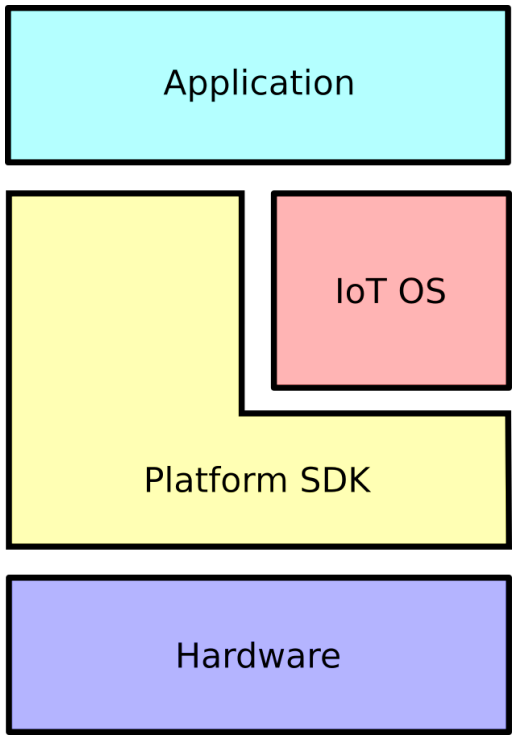
What will we look at?

What will we look at?



less than 1MB of RAM, no MMU, often 16/32 Bit

What will we look at?



Between 50 and 300 kLoC without platform drivers

History

Name	Issues reported
RIOT OS Core	3
Wakaama	3
Tiny-ASN1	5
SPIFFS	4
TinyDTLS	7
CCN-Lite	>15
OONF	3

OS Programming in 2018...

- We have secure languages
- We have extensive test suites and fuzzing
- We have a lot of hardening techniques
- We have static and dynamic analysis tools

Hardening

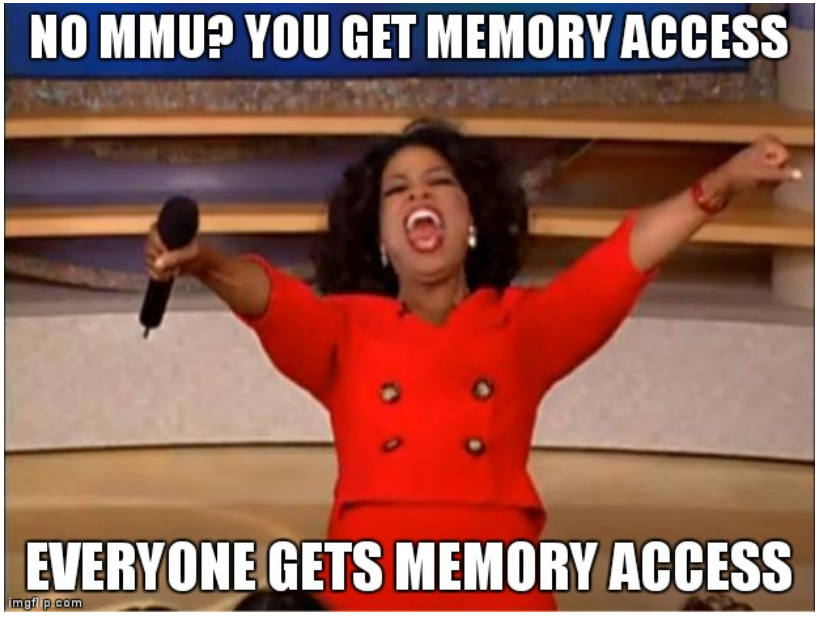
	Audits	Static Analysis	Fuzzing	Hardening
Contiki	No	No	No	No
MyNewt	No	Coverity	1 (lwIP)	No
RIOT	No	Clang?	1 (SPIFFS)	Stack Prot.
Zephyr	No	Coverity	No	Stack Prot.

Memory Management Unit

- Manages access to memory
- Allows to set access restrictions
- Without it, no separation of user space and kernel space
- Additionally, no users, no sandboxes...

MMU Support

- Contiki - no
- MyNewt - no
- RIOT - no
- Zephyr - on X86



Crypto

- How to secure IoT?
- Sprinkle some magic cryptographic dust on it...

"The security functionality in Zephyr hinges mainly on the inclusion of cryptographic algorithms, and on its monolithic system design."

Randomness - TinyDTLS

```
1  dtls_ticks(&now);
2  #if (defined(WITH_CONTIKI) || defined(RIOT_VERSION))
3  /* FIXME: need something better to init PRNG here */
4  dtls_prng_init(now);
5  #else /* WITH_CONTIKI */
```

Randomness - TinyDTLS

- Uses ***rand()***
- `man 3 rand`: "...so this function will be a weak pseudo-random generator."

```
1  dtls_prng(unsigned char *buf,
   ↪  size_t len) {
2    while (len--)
3      *buf++ = rand() & 0xFF;
4    return 1;
5  }
```

Randomness - Contiki

- No entropy pool
- ***random_rand()*** to read CPU HWRNGs directly
- Fallback to ***gcc rand()***
- nRF51 can fail/hang forever -> not handled
- CC13xx/CC26xx might return 0, always < 0xFFFF

Randomness - MyNewt

- Some pools, entropy is never added
- Some unused HWRNG
- *ble_hs_hci_util_rand()* and *HAL_RNG_GenerateRandomNumber()* can fail

Randomness - RIOT

- No entropy pool
- Several PRNG
- Code to read CPU HWRNGs directly - not used...

Randomness - Zephyr

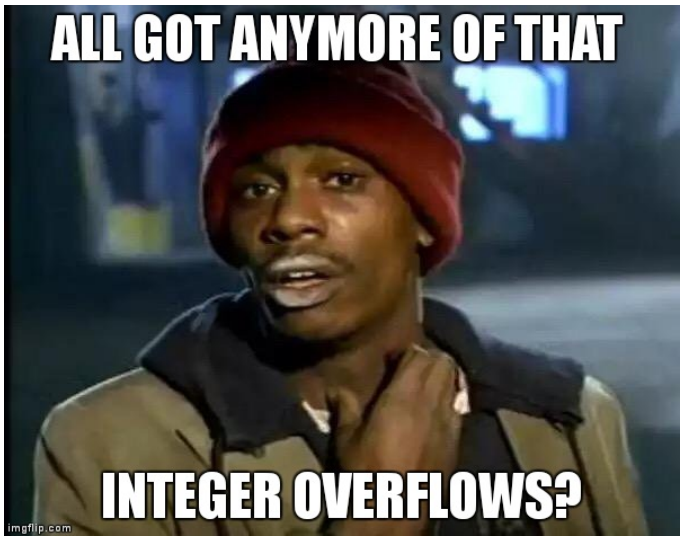
- No Pools
- Driver to use HWRNG
- Fallback to time-based and PRNG
- xoroshiro128 is not cryptographically secure
- mbedtls has its own pool

Randomness - What would we want?

- Think */dev/urandom*
- A pool, that can not fail!
- Possible input from multiple sources
- Everything provided by the OS
- Cryptographically secure PRNG
- Depending on the device, you might need a */dev/random* equivalent for long lived keys

Allocators

- *malloc()* might be a bad idea depending on your threat model ...
- External non-IoT code likes *malloc()*
- Usually fallback to platform SDK



Allocators - Contiki

- Mostly avoids ***malloc()*** and friends!
- Uses memb memory block allocator and mmem managed memory allocator

Allocators - Contiki memb

- Commonly used in Contiki
- Metadata out of band (in an array)
- Always checks all blocks O(n) in ***alloc()/free()***
- Continuous block of memory, no guard pages/overflow protection

Allocators - Contiki mmem

- Only used by AVR ELF loader
- Metadata out of band (linked list)
- Continuous block of memory, no guard pages/overflow protection
- Overwrite of block always hits used memory except for the last block!

Allocators - MyNewt

- Double linked allocated and free area list
- Metadata inline
- Double free check for *DEBUG_MALLOC*
- No Safe Unlinking
- No Guard Checks

Allocators - MyNewt

- Classic Integer Overflow
- Fixed in glibc and others in 2002
- *baselibc/src/calloc.c*

```
1  /* FIXME: This should look
2  ⌋
     ↪      for multiplication overflow
3
4  void *calloc(size_t nmemb,
     ↪   size_t size)
5  {
6          void *ptr;
7
8          size *= nmemb;
9          ptr = malloc(size);
```

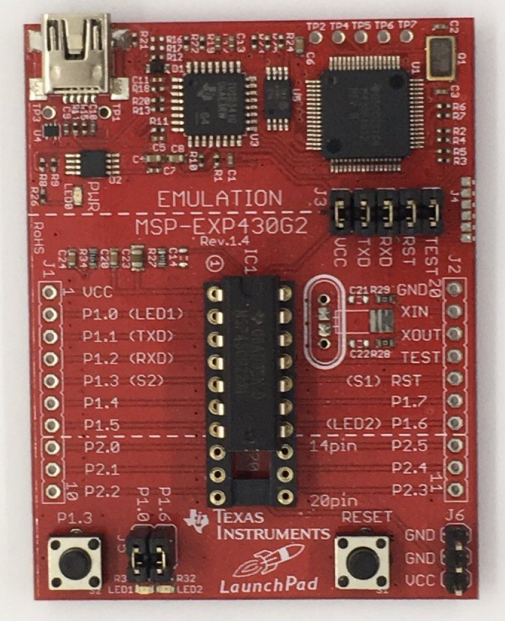
More Bugs - Apache MyNewt

- *baselibc/src/-malloc.c*
- Integers can overflow from additions as well...

```
1  void *malloc(size_t size) {
2  ...
3    if (size == 0)
4      return NULL;
5
6    /* Add the obligatory arena
7       header, and round up */
8    size = (size + 2 * sizeof(struct
       ↪  arena_header) - 1) &
       ↪  ARENA_SIZE_MASK;
```

Allocators - RIOT

- Just one by MSP430
- Wrapper for ***sbrk()***
- Double Free Protection...
  free() is not implemented
  ;-)



Allocators - RIOT

- Common pattern?
- Used by MSP430 - *sys/oneway-malloc/oneway-malloc.c*
- Reported, not yet fixed...

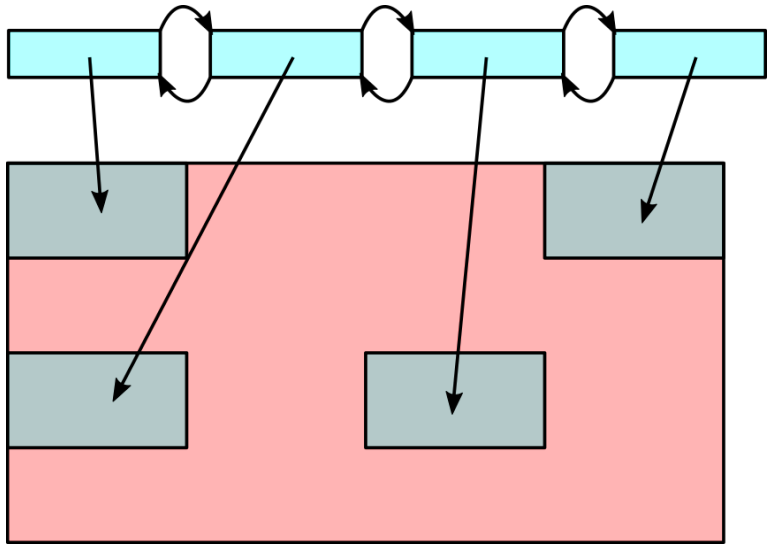
```
1  void __attribute__((weak))
   ↪  *calloc(size_t size, size_t
   ↪  cnt)
2  {
3    void *mem = malloc(size * cnt);
4    if (mem) {
5      memset(mem, 0, size * cnt);
6    }
7    return mem;
8  }
```

Allocators - Zephyr

- Slab for fixed sized objects
- Heap for variable sized objects

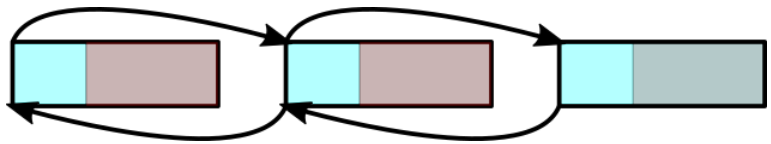
Allocators - Zephyr Slab

- Continuous block of memory
- Free chunks linked in list
- Allocated blocks not tracked by allocator
- No double free detection
- No guards



Allocators - Zephyr Heap

- Metadata inline
- No double free protection
- No safe unlinking



Allocators - Zephyr

- *kernel/mempool.c*
- Looks familiar?

```
1  void *k_calloc(size_t nmemb,
    ↪  size_t size) {
2  ...
3  #ifdef CONFIG_ASSERT
4    __ASSERT(!__builtin_mul_overflow
      ↪  (nmemb, size,
      ↪  &bounds),
5      "requested size overflow");
6  #else
7    bounds = nmemb * size;
8  #endif
```

Allocators - Zephyr

- *kernel/mempool.c*
- yawn...

```
1  void *k_malloc(size_t size) {
2    struct k_mem_block block;
3
4    /* get a block large enough to
5     * hold an initial (hidden)
6     * block descriptor, as well
7     * as the space the caller
8     * requested    */
9    size += sizeof(struct
       ↪  k_mem_block);
```

Allocators

- Always fun to look at, none of them hardened
- Browsers in 2018 bring their own hardened allocators...
- Some more bugs in there, not all ***realloc()*** implementations POSIX compliant - Original block should not be touched/freed on failure

What else to attack?

- Depends on your goal ... but ... Parsers :-)
- Want to get *root* to install your own software? Filesystems
- Want to abuse the device? Network
- "Borrowed code" - Code from other projects, where stuff might already be fixed

More Bugs - Contiki

- *apps/webbrowser/-www.c*
- Easily found
- Only overwrites the current web site displayed

```
1  static char
     ↪  url[WWW_CONF_MAX_URLLEN + 1];
2  static void
3  set_link(char *link)
4  {
5  ...
6    } else if(*link == ISO_slash &&
7            *(link + 1) == ISO_slash) {
8  ...
9      strncpy(&url[5], link,
         ↪  WWW_CONF_MAX_URLLEN);
```

More Bugs - Zephyr

- *subsys/net/lib/-lwm2m/-lwm2m_engine.c*
- ***snprintk()*** behaves like ***snprintf()***
- Returns the amount of bytes it would print
- Other cases check just < 0

```
1  static char buf[32];
2  ...
3  for (i = 0; i < tkl; i++) {
4    pos += snprintk(&buf[pos], 31 -
       ↪  pos, "%x", token[i]);
5  }
```

More Bugs - Contiki

- *z1-websense.c*
- Same class as above

```
1  #define ADD(...) do { \
2    blen += snprintf(&buf[blen], \
3  ⌋
    ↪      sizeof(buf) - blen, __VA_ARGS__)
4  } while(0)
```

Conclusions

Popular Open Source IoT OS are

- Written in C :(
- Not hardened
- Buggy in the allocators
- Not providing good randomness infrastructure

Thanks

- Q & A
- x41-dsec.de/reports/smells.pdf
- eric.sesterhenn@x41-dsec.de
- Sorry no Twitter... stalk me on LinkedIn if you must ;-)