

# AURIX™ Knowledge Lab 2021

## Security Aspects of Static Code Analysis



**Frank Büchner**  
Principal Engineer  
Software Quality

[frank.buechner@hitex.de](mailto:frank.buechner@hitex.de)



1. Safety vs. Security

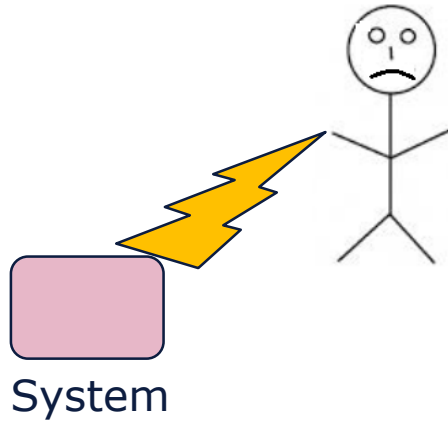
2. Security

3. Coding Guidelines for the Security of Software

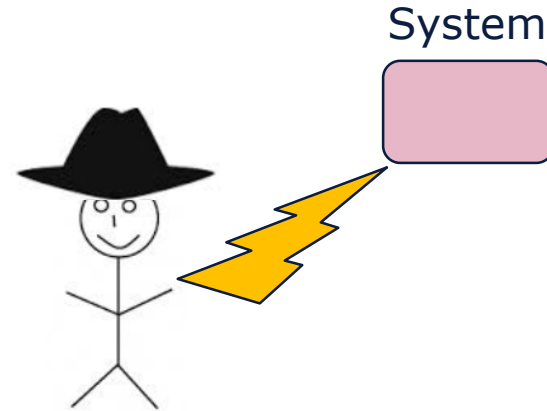
4. How Can Static Analysis Tools Help?

5. Conclusion

# Safety vs. Security



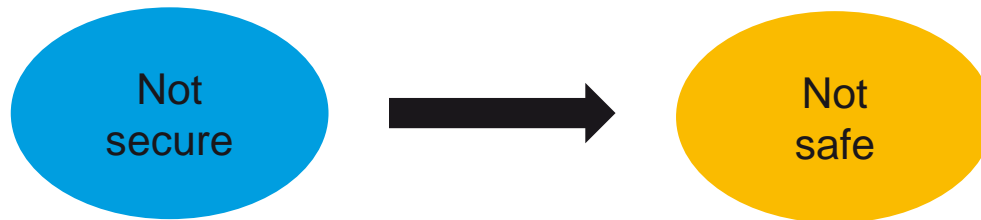
Safety



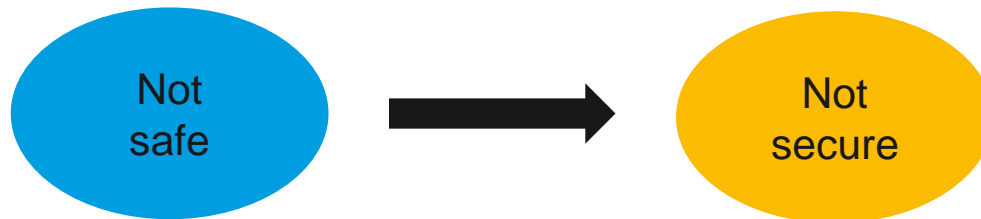
Security

# Safety vs. Security

- If a system is not secure, it cannot be safe



- If a system is not safe, can it be secure?



1. Safety vs. Security



2. Security

3. Coding Guidelines for the Security of Software

4. How Can Static Analysis Tools Help?

5. Conclusion

# Goals of Security

- Confidentiality

- Integrity

- Availability

- Secure boot / root of trust
- Secure firmware update
- Cryptography
- Authentication
- Appropriate passwords
- Social engineering attacks
- ...
- Secure software (firmware)

1. Safety vs. Security

2. Security



3. Coding Guidelines for the Security of Software

4. How Can Static Analysis Tools Help?

5. Conclusion

## ■ Coding guidelines related to “security”

### □ ISO/IEC TS 17961:2013

- C **secure** coding rules
- Canadian Standards Association (CSA)
- 64 guidelines



## ■ Coding guidelines related to “security”



### □ SEI CERT C Coding Standard

- SEI = Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA
- CERT = Computer Emergency Response Team
- “This standard provides rules for **secure** coding in the C programming language.”
- 99 guidelines (edition 2016)
- <https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard>

## ■ Databases related to “security”

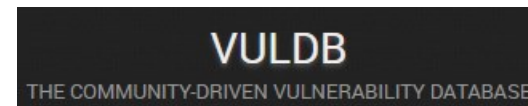
### □ Common Weakness Enumeration (CWE)

- Common language
- Measuring stick for **security** tools
- <https://cwe.mitre.org>



### □ VULDB

- <https://vuldb.com>



- Guidelines related to “safety” also tackle “security”
  - MISRA C:2012 (mainly perceived “safety-related”)
    - 2016: Amendment 1: 14 new guidelines for security
    - 2018: Addendum 2: Coverage against ISO/IEC 17961 “C secure”
      - All 64 guidelines are covered
    - 2018: Addendum 3: Coverage against CERT C (2016)
      - Of the 99 guidelines are 80 (more or less) covered; 15 out of scope; 4 uncovered.



Strong relation of safety and security

1. Safety vs. Security

2. Security

3. Coding Guidelines for the Security of Software



4. How Can Static Analysis Tools Help?

5. Conclusion

- Static analysis tools can automatically check for violations of coding guidelines
- However
  - Rules can be undecidable (halting theorem)
    - False negatives
    - False positives
  - Soundness: Are false negatives possible?

## ■ Example “Possible NULL pointer dereferencing”

```
13      str = (char *) malloc(15);
```

**EMC3.D4.7** the return value for call to function `malloc(size\_t)' is not tested for being equal to the null pointer

```
14      strcpy(str, "something");
```

```
15
```



Defect! Might crash. Affects availability.

CERT: EXP34-C. Do not dereference null pointers

17961: 5.14 [nullref]

CWE-476: NULL Pointer Deference

## ■ Example "Buffer overflow"

```
11      char s[] = "The name is: X";  
12      const char *name = "Frank";  
13      strcpy(s + 13, name);
```



≡B.BUGFIND.unix.cstring.OutOfBounds string copy function overflows destination buffer

```
14
```

Defect! Classic security issue.

CERT: ARR30-C: Out-of-bounds ... array subscripts. STR31-C: ... sufficient space ...

17961: 5.22 [invptr]

CWE-120: Classic buffer overflow

## ■ Example “Tainted input”



```
19      fgets(input_buf, sizeof(input_buf), fp);
```

EMC3A1.D4.14 the value written to the reference to variable `input\_buf` at #1 in call to function `fgets(char\*, int, FILE\*)' is not checked as expected

```
20      system(input_buf);
```

```
21
```

Defect? Definitive security issue.

CERT: ?

17961: 5.37 [taintstrcpy]? 4.14: Tainted source include strings produced by fgets()

CWE-20: Improper Input Validation

## ■ Example "Tainted input"

```
// input_buf checked
fgets(input_buf, sizeof(input_buf), fp);
if (check_string_for_system(input_buf))
{
    system(input_buf);
}
```



No violation of MC3A1.D14.4 any more!

```
-config=MC3R1.D4.14,+argument_post_check={return_val, "node(if_stmt)&&child(cond,call(decl(^check_string_.*$)))"}
```

## ■ Example “Hard-coded password”

```
char current_pw[] = "123456";

// set password
void set_pw(void)
{
    // Not implemented yet. We implement a proper password
    // manager if we have time. Use default password for now.
}

void main(void)
{
    // Force user to change initial password
    set_pw();

    /* ... */
}
```

No defect, but a definitive security issue.

CWE-259: Use of Hard-coded password

1. Safety vs. Security
2. Security
3. Coding Guidelines for the Security of Software
4. How Can Static Analysis Tools Help?



5. Conclusion

- Static analysis tools help, but additional measures are needed
  - Requirements
  - Review
- Sound checkers are important
- Too many false positives are a problem

# Thank you for your attention!

## Any questions?



**Frank Büchner**  
Principal Engineer  
Software Quality

[frank.buechner@hitex.de](mailto:frank.buechner@hitex.de)

[www.hitex.com/eclair](http://www.hitex.com/eclair)