Vulnerabilities in Safety, Security, and Privacy

Commonalities, Differences and Useful Sources of Information

Erhard Plödereder
University of Stuttgart, Germany

The Problem

"Producing better and safer software is much too expensive. The costs do not amortize sufficiently over our product fleets."

Anonymous manager in the automotive industry (ca. 2010)

But: The real problem is not the red statement! ….

The Case "Toyota"

Up to 2014 the NASA space shuttle software was among the most expensive software in the universe: about 1000 $ per Line of Code (LoC) with error rates of 0,0025 errors/kLoC.
(Source: F. Pickhard, ETAS, 2014, based on "201 Principles of Software Development", Alan Davis, 1995)

After 2014 the Toyota *Electronic Throttle Control System, Intelligent* (ETCS-i) holds/held the world record:

- ca. 100 $/LoC development costs (my – irrelevant - guess)
- 1.600.000.000 $ class action settlement
- 1.200.000.000 $ punitive damages
- 3.000.000.000 $ recalls, probes, lost sales, etc.
- With about 1,000,000 LOC in ETCS-i , these costs add up to

more than 5000 $ per line of code !

(Sources: various Internet sources (Forbes, Wall Street Journal, Toyota) and in particular the court records of Koopman vs. Toyota)

Today's Situation (in a Nut Shell)

Critical software in cars has requirements for the degree of ***Reliability and Safety*** *similar to the software for controlling nuclear reactors (and higher than for space crafts).*

By opening up cars for X-2-C communication, the demands on the ***Security*** *of the car-based systems grew considerably.*

Especially in Europe, there is customer demand for securing the ***Privacy*** *of personal data transmitted by C-2-X communication.*

however …

A few metrics on lines of code:

Avionics system of the F-22 Raptor: 1,7 MLoC

Onboard systems of F-35 Joint Strike Fighter: 5,7 MLoC

Avionics and onboard support systems (without infotainment) of the Boeing 787 Dreamliner: 6,5 MLoC

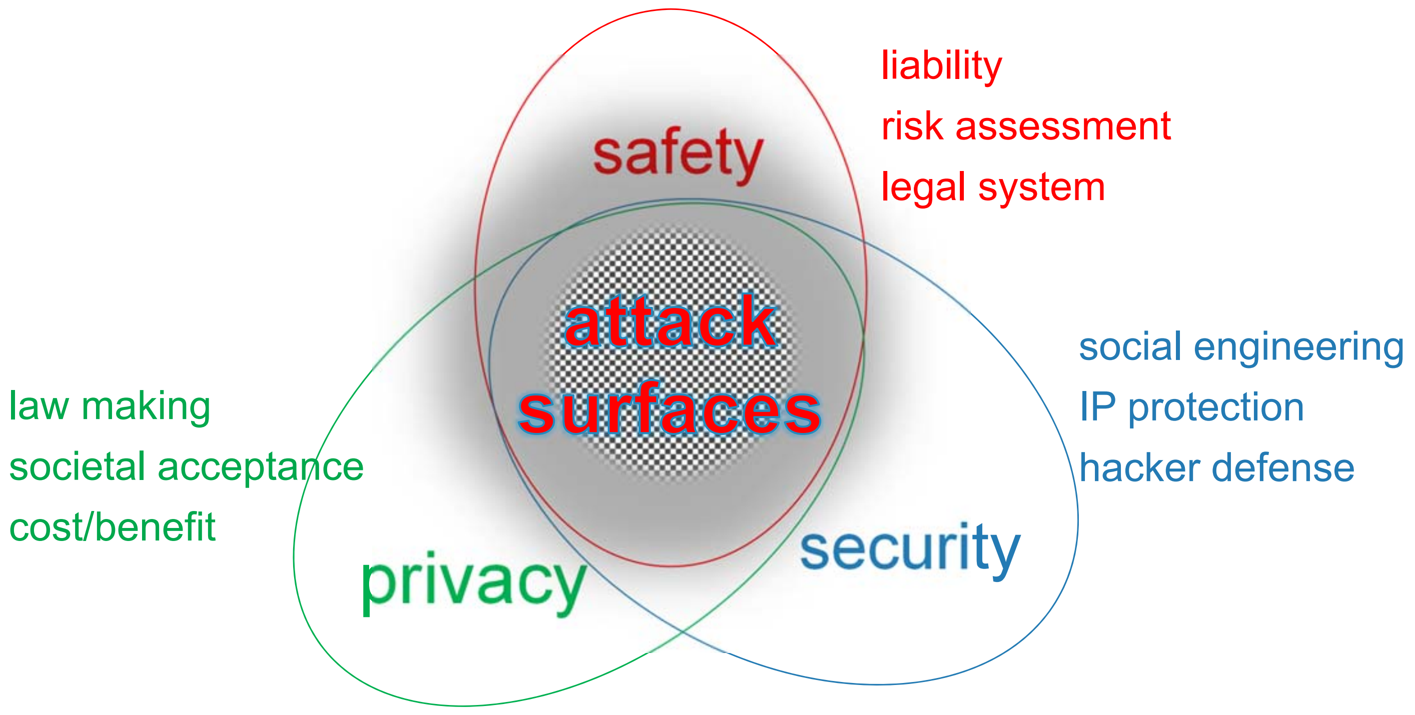
Radio and navigation in a Daimler S-class car: 20 MLoC
source: A. Katzenbach, Daimler AG

German luxury car, sum total: 100 MLoC
source: M. Broy, TU Munich

?? 1 car is more complicated than 10 Dreamliner ??

source: IEEE Spectrum (online): **This Car Runs on Code**, Robert N. Charette, 1. Feb. 2009

Connections and Concerns



Attack Surfaces

„δῶς μοι πᾶ στῶ καὶ τὰν γᾶν κινάσω"
("Give me a place to stand on, and I will move the Earth.*“)*
Archimedes, according to Pappos in *Collections, Book VIII*, and Wikipedia

turns into

*"*Give me access to one of the ECUs, and I will own your car."
car hacker of the 21. century, in *Annals of the ConnectedDrive*

Postscript: *"*A communication bus is likely to suffice as well*."*

Primary Attack Surfaces

- *The human, whether naive or bribed (leaking private keys or passwords, negligence, unauthorized access to company computers, selling of company IP)*
- *spoofing communication*
- *accidental (=> safety, reliability) or malevolent (=> safety, privacy) exploitation of requirements, design or coding errors and weaknesses in soft- or hardware*

An Open Barn Door?

```
char *copy(size_t n, const char *a) {
      if (n == 0) return NULL;
      char *p = (char *) malloc(n);
      if (p == NULL) return NULL;
      for (int i = 0; i < n; ++i) p[i] = *a++;
      return p;
}
```

Ist this code o.k. or not?

Code reviewer is likely to say: √ **Tester is likely to say: √**

source: example (but not conclusions) copied from presentations by Robert Seacord, formerly CERT/SEI

An Open Barn Door!

```
char *copy(size_t n, const char *a) {
      if (n == 0) return NULL;
      char *p = (char *) malloc(n);
      if (p == NULL) return NULL;
      for (int i = 0; i < n; ++i) p[i] = *a++;
      return p;
}
```

As long as `copy` is called with an n equal to the length of a (and not excessively large) the code operates correctly. The "vulnerability" is therefore unlikely to be discovered during functional testing.

However, if n "lies", the entire memory can be read. Safety is (almost) unaffected, but Security and Privacy are severely compromised ("Heartbleed"). If the assignment is inverted, arbitrarily large memory can be overwritten: both Safety and Security are affected.

With n close to maxint, memory gets tight and the loop runs for a looooong time (DoS attack).

Mutual Dependence

- If Security cannot guarantee the integrity of code and data, there can be no guarantee of Safety. A viral attack can kill otherwise completely reliable code (and people).
- If Reliability and Safety cannot prevent the malevolent exploitation of "vulnerabilities" and "weaknesses", one cannot possibly presume any Security.
- The malevolent theft of private data becomes possible when the implemented safeguards to protect against unauthorized access can be subverted by the means above.

Vulnerabilities

ISO TR 24772:2013 defines "Vulnerability" as follows:

"All programming languages contain constructs that are incompletely specified, exhibit undefined behaviour, are implementation-dependent, or are difficult to use correctly. The use of those constructs may therefore give rise to ***vulnerabilities*****,** as a result of which, software programs can execute differently than intended by the writer. In some cases, these vulnerabilities can compromise the safety of a system or be exploited by attackers to compromise the security or privacy of a system."

Vulnerabilities

The NIST National Vulnerability Database (NVD) defines a vulnerability as:
"A weakness in the computational logic (e.g., code) found in software and hardware components that, when exploited, results in a negative impact to confidentiality, integrity, or availability. Mitigation of the vulnerabilities in this context typically involves coding changes, but could also include specification changes or even specification deprecations (e.g., removal of affected protocols or functionality in their entirety)."

(Some other definitions apply the term "vulnerability" only in connection with security und privacy in the context of malevolent attacks.)

Vulnerabilities and Weaknesses

How does one learn about them and their dangerous consequences, and how does one avoid them?

In the following slides, I provide several sources, describe their contents, and give links to retrieve the information:

- MISRA
- CWE – Mitre
- CERT/CC
- ISO Standards
- JSF Coding Standard
- C++ Core Guidelines
- Company standards

MISRA-C (1998, 2004, 2012, 2016)

MISRA = The Motor Industry Software Reliability Association

*www.**misra**.org.uk*

Guidelines for the Use of the C Language in Critical Systems

- Rules (1998: 127, 2004: 144 rules) constraining C constructs in contexts of known maintenance or reliability problems.
- The majority of these rules (but not all!) are checkable by a variety of available static analysis tools for C programs.
- MISRA-C rules can be seen as a binding standard (state-of-the-practice) for QA in the automotive sector. They were prominently mentioned in the Toyota court case in the USA.

MISRA-C (1998)

Rule 13: "The basic types of char, int, short, long, float and double should not be used, but specific length equivalents should be typedef'd for the specific compiler, and these type names used in the code." *

Rule 25: "An identifier with external linkage shall have exactly one external definition." *

* Source: "**A Comparison of MISRA C Testing Tools",** presented at the MISRA C Forum 18 October 2001

MISRA-C

"The Camry ETCS code was found to have 11,000 global variables. …. Using the Cyclomatic Complexity metric, 67 functions were rated untestable (meaning they scored more than 50). The throttle angle function scored more than 100 (unmaintainable).

Toyota loosely followed the widely adopted MISRA-C coding rules but Barr's group found 80,000 rule violations. Toyota's own internal standards make use of only 11 MISRA-C rules, and five of those were violated in the actual code."

(Source: Michael Dunn, "**Toyota's killer firmware: Bad design and its consequences**" in EDN Network, 28.10.2013)

MISRA-C++ (2008)

- An extension of MISRA-C with C++ rules
- Unlike the widely distributed MISRA-C rules, the C++ rules are proprietary; they are not freely available and require licensing by MISRA
- … consequently I cannot cite its contents or discuss tools.

CWE (V 3.1)

CWE = Common Weakness Enumeration
https://cwe.mitre.org/

- A collection (995 entries on 10.6.18) of safety or security problems encountered in real-world systems, combined with hints for avoiding or mitigating them
- Initially mostly dealing with safety issues, but for several years now strongly focused on security breaches
- The collection is actively maintained and catalogued
- Includes Top 10 and Top 25 lists from different communities

CWE-365: Race Condition in Switch

Description

Summary: The code contains a switch statement in which the switched variable can be modified while the switch is still executing, resulting in unexpected behavior.

Extended Description: …

Time of Introduction: Implementation

Applicable Platforms: C, C++, Java, C#

Common Consequences: …

Likelihood of Exploit: Medium

Demonstrative Examples: ….

Potential Mitigations: Variables that may be subject to race conditions should be locked before the switch statement starts and only unlocked after the statement ends.

Quelle: online CWE at https:cwe.mitre.org

CERT Publications

CERT = *Computer Emergency Response Team*, located at the SEI of Carnegie Mellon University

https://www.securecoding.cert.org

- The CERT C Secure Coding Standard, Second Edition (Addison-Wesley, 2014)
- Secure Coding in C and C++, Second Edition (Addison-Wesley, 2013)
- The CERT Oracle Secure Coding Standard for Java (Addison-Wesley, 2011)
- AndroidTM Secure Coding Standard (online)
- CERT PERL Coding Standard (online)

CERT C Secure Coding Standard

MEM36-C. Do not modify the alignment of objects by calling realloc()

Do not invoke realloc() to modify the size of allocated objects that have stricter alignment requirements than those guaranteed by malloc(). Storage allocated by a call to the standard aligned_alloc() function, for example, can have stricter than normal alignment requirements. The C standard requires only that a pointer returned by realloc() be suitably aligned so that it may be assigned to a pointer to any type of object with a fundamental alignment requirement.

Noncompliant Code Example …

Compliant Solutions ….

Risk Assessment: Improper alignment can lead to arbitrary memory locations being accessed and written to.

Recommen-dation	Severity	Likelihood	Remediation Cost	Priority	Level
MEM36-C	Low	Probable	High	**P2**	**L3**

Automated Detection: <<analysis tools that detect this problem>>

ISO-WG14 Publications

- JTC1/SC22 WG14 ("C"): ISO/IEC TS 17961 C Secure Coding Rules (TS = Technical Specification)
- Coding rules for C formulated as rules checkable by analysers, e.g., (Source: draft TS 17961; 5.30, very close to 5.31, the official TS:2013)

5.30 Passing a non-null-terminated string to a library function [nonnullstr]
Rule
Passing a string or wide string that is not null-terminated to such a function shall be diagnosed.
Rationale
Many library functions accept a string or wide string argument with the constraint that the string they receive is properly null-terminated. Passing a string or wide string that is not null-terminated to such a function can result in accessing memory that is outside the bounds of the string.
Example(s) ….

ISO-WG23 Publikationen

JTC1/SC22 WG23 ("OWG"): TR 24772-1 (Ed. 3) Guidance to Avoiding Vulnerabilities in Programming Languages
(TR = Technical Report)

97 vulnerabilities at coding level (64) or design or environment level (33).

The TR Part 1 contains language-independent descriptions of the vulnerabilities, their consequences upon enactment, possibilities for malicious exploitation, and rules for avoiding or mitigating the vulnerabilities.

TR 24772 Part 2-10 (Ada, C, C++, Fortran, PHP, Python, Ruby, Spark) contain the matching specifics for the particular language and its means for countering the vulnerabilities. *(presently in revision for Ed. 3 of Part 1)*

Excerpt from TR 24772-1 (V3; 2018)

6.15 Arithmetic Wrap-around Error [FIF]
6.15.1 Description of application vulnerability
Wrap-around errors can occur whenever a value is incremented past the maximum or decremented past the minimum value representable in its type and, depending upon • whether the type is signed or unsigned, • the specification of the language semantics and/or • implementation choices, "wraps around" to an unexpected value. …
6.15.2 Cross reference
CWE: 128. Wrap-around Error 190. Integer Overflow or Wraparound , JSF AV Rules: 164 and 15 MISRA C 2012: 7.2, 10.1, 10.3, 10.4, 10.6, 10.7, and 12.4 , MISRA C++ 2008: 2-13-3, 5-0-3 to 5-0-10, and 5-19-1 CERT C guidelines: INT30-C, INT32-C, and INT34-C
6.15.3 Mechanism of failure …
Wrap-around often generates an unexpected negative value; this unexpected value may cause a loop to continue for a long time …or an array bounds violation. A wrap-around can trigger buffer overflows that can be used to execute arbitrary code.
6.15.4 Applicable language characteristics …
6.15.5 Avoiding the vulnerability or mitigating its effects …
6.15.6 Implications for standardization …

Excerpt from TR 24772-2, Ada Part

6.15 Arithmetic Wrap-around Error [FIF]
With the exception of unsafe programming (see 4 Language Concepts), this vulnerability is not applicable to Ada as wrap-around arithmetic in Ada is limited to modular types. Arithmetic operations on such types use modulo arithmetic, and thus no such operation can create an invalid value of the type.
For non-modular arithmetic, Ada raises the predefined exception Constraint_Error whenever a wrap-around occurs but implementations are allowed to refrain from doing so when a correct final value is obtained. In Ada there is no confusion between logical and arithmetic shifts.

TR 24772:2013 (V2), Python Annex

E.16 Arithmetic Wrap-around Error [FIF]
E.16.1 Applicability to language
Operations on integers in Python cannot cause wrap-around errors because integers have no maximum size other than what the memory resources of the system can accommodate.
Normally the OverflowError exception is raised for floating point wrap-around errors but, for implementations of Python written in C, exception handling for floating point operations cannot be assumed to catch this type of error because they are not standardized in the underlying C language. Because of this, most floating point operations cannot be depended on to raise this exception.
E.16.2 Guidance to language users
• Be cognizant that most arithmetic and bit manipulation operations on non-integers have the potential for undetected wrap-around errors.
• Avoid using floating point or decimal variables for loop control but if you must use these types then bound the loop structures so as to not exceed the maximum or minimum possible values for the loop control variables.
• Test the implementation that you are using to see if exceptions are raised for floating point operations and if they are then use exception handling to catch and handle wrap-around errors.

JSF Coding Standards for C++

- JSF = Joint Strike Fighter
- 201 rules, largely developed by Bjarne Stroustrup, Designer of C++
- Partly relating to development process, mainly style guides or language restrictions with the associated technical reasons; many rules are not (easily) checkable automatically
- Available at
  http://www.stroustrup.com/JSF-AV-rules.pdf (Dec. 2005)

JSF Coding Standards for C++

AV Rule 77 A copy constructor **shall** copy all data members and bases that affect the class invariant (a data element representing a cache, for example, would not need to be copied).
Rationale: Ensure data members and bases are properly handled when an object is copied. See AV Rule 77 in Appendix A for additional details.

AV Rule 77.1 The definition of a member function **shall not** contain default arguments that produce a signature identical to that of the implicitly-declared copy constructor for the corresponding class/structure.
Rationale: Compilers are not required to diagnose this ambiguity. See AV Rule 77.1 in Appendix A for additional details.

AV Rule 78 All base classes with a virtual function **shall** define a virtual destructor.
Rationale: Prevent undefined behavior. If an application attempts to delete a derived class object through a base class pointer, the result is undefined if the base class's destructor is non-virtual.

C++ Core Guidelines

An ongoing project by Bjarne Stroustrup and Herb Sutter to collect meaningful guidelines for improving code quality.
https://isocpp.github.io/CppCoreGuidelines

The guidelines include but are not targeted only to vulnerability avoidance. Checkability is an important criterion in composing these rules.

Example:

Enum.1: Prefer enumerations over macros
Reason
Macros do not obey scope and type rules. Also, macro names are removed during preprocessing and so usually don't appear in tools like debuggers.
Example
….<< good and bad code>>
Enforcement
Flag macros that define integer values.

Company Standards

Companies like Microsoft, Google, or consortia such as AutoSar publish their Coding Guidelines (suppliers are usually bound by these guidelines), e.g.,

- Microsoft's "Security Development Lifecycle (SDL) Banned Function Calls" prohibits the use of about 200 functions of the C standard library and cites safer alternatives.

  https://msdn.microsoft.com/en-us/library/bb288454.aspx

- A recent entry is the AutoSar Coding Standard (2017): *Guidelines for the use of the C++14 language in critical and safety-related systems*

  https://www.autosar.org/.../AUTOSAR_RS_CPP14Guidelines.pdf

Process Standards

A series of important standards dealing mainly with the processes in software design and development, exist but have not been included here, e.g.

ISO 61508, ISO 26262, CENELEC EN 50126, DoD-178B und C and others.

Resume

- Reputable sources exist that can be consulted to derive rules and guidelines for the design and development of systems in which safety, security or privacy are of fundamental importance.
- There is no "one size fits all" set of rules, since, for example, different SIL/ASIL levels are to be taken into consideration. What may be acceptable at level 2 might be utterly forbidden at level 3. Moreover, there is significant influence by the programming language(s) used.
- **Guidelines and, in particular, their continuous checking are an essential ingredient on the way to safer software.**
- ***Problem #1: which of the >>1000 rules are important for my project? (Please do not reuse the guidelines invented in the days of assembler programing!)***

Reprise (Reality Check)

"…
Toyota loosely followed the widely adopted MISRA-C coding rules but Barr's group found 80,000 rule violations. Toyota's own internal standards make use of only 11 MISRA-C rules, and five of those were violated in the actual code."
(Source: Michael Dunn, "**Toyota's killer firmware: Bad design and its consequences**" in EDN Network, 28.10.2013)

Problem 2: *… and how do I convince my developers to heed the rules imposed?*

My personal advice …

- Continuous integration to catch problems early …
- … combined with automated vulnerability detection
- Choose your static analysis tools wisely
    - which rules are checked?
    - good diagnostics!
    - blessings and tracking possible
    - configurable
- Be lenient about fixing vulnerabilities … (initially)
- Run short sprints for vulnerability elimination when their number exceeds a threshold or when release dates are near
- Above all: create a common understanding by your team why vulnerabilities need to disappear from the code

Borrowing from the German Catalogue of Traffic Violations 2018 …

• *exceding speed limit by up to 21 km*	*70€*
• *running a red light (no endangerment)*	*90€*
• *running a red light (with endangerment)*	*200€*
• *illegal road race*	*jail up to 2 years*
• *negligence causing accidental death*	*criminal offense*
• *…*	
• *…. (from the catalogue 2025) ??:*	
• *Buffer overflow (no endangerment)*	*1000€ (programmer)*
• *Buffer overflow (no endangerment)*	*10000€ (project leader)*
• *Buffer overflow (with endangerment)*	*....t.b.d.*
• *Buffer overflow (causing a death)*	*....criminal offense*