Utilizing security methods of FLOSS GPOS for safety

Nicholas Mc Guire
Distributed & Embedded Systems Lab
Lanzhou, China
safety@osadl.org, mcguire@lzu.edu.cn

Security for Safety

Reuse of security methods can potentially cover the entire life-cycle of systems.

- Why look at security at all ?
- High-level concept and Standards reuse
- Security vs Safety - differences and overlap
- security methods potentially suitable
- field data (though de-scoped for this talk)

This is not to state that one can simply use security methods to provide safety - but one can use security methods to mitigate some threats and enhance safety related systems.

Safety is ”discovering” its security demands

- Fail safe is quite often a (dangerous) illusion
- Safety and security demands are often viewed as being opposing - this is only partially true
- Safety by secrecy is unfortunately still the norm - Security by obscurity was abandoned more than 20 years ago !
- If security and safety is integrated in the right way - we believe that safety can benefit from a lot of the established security methods

Safety and security are attending to two problem domains - but our claim is that the overlap is far greater than many anticipate - reusing successful security methods for safety is feasible and, as I believe, advisable.

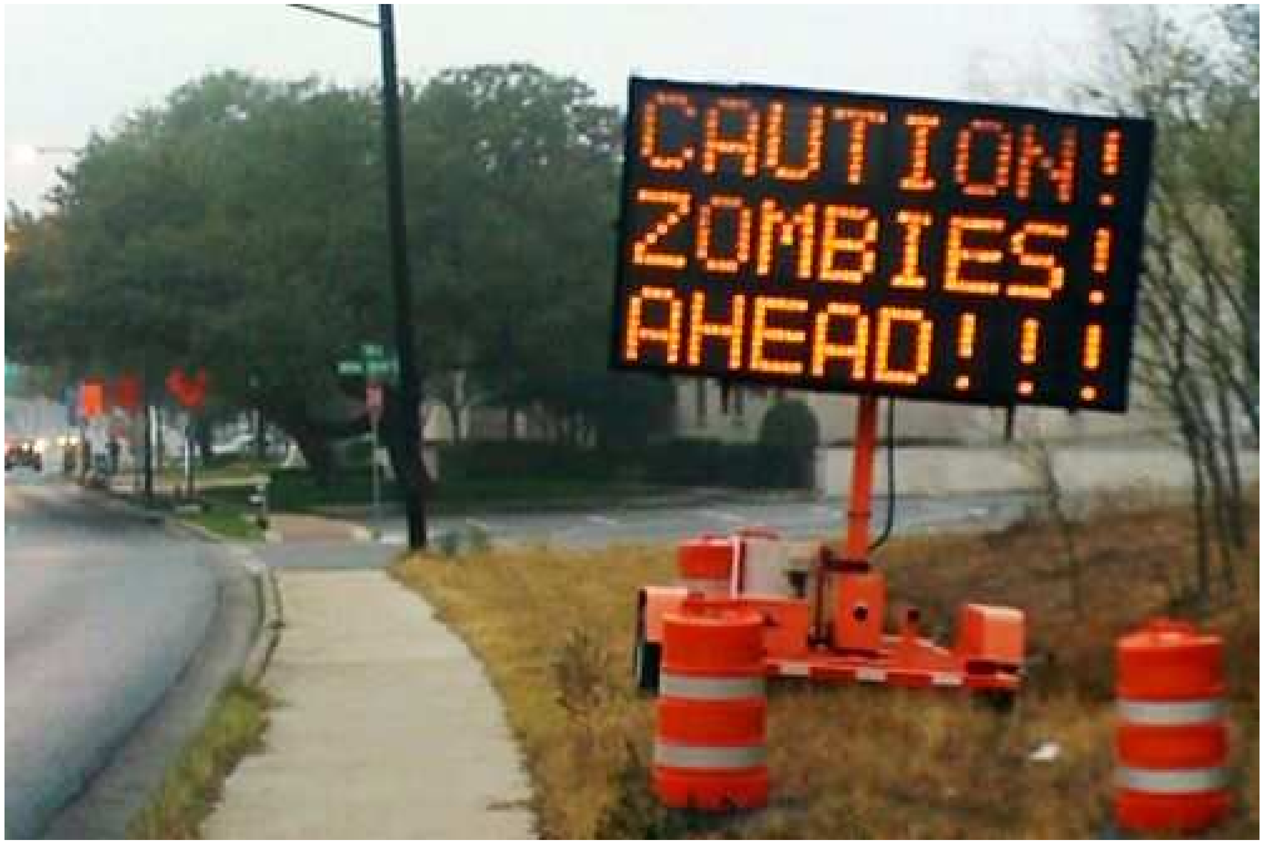
Security issues in 61508

Security has traditionally been excluded from functional safety considerations - this is (maybe) now changing. The two cases to be distinguished are:

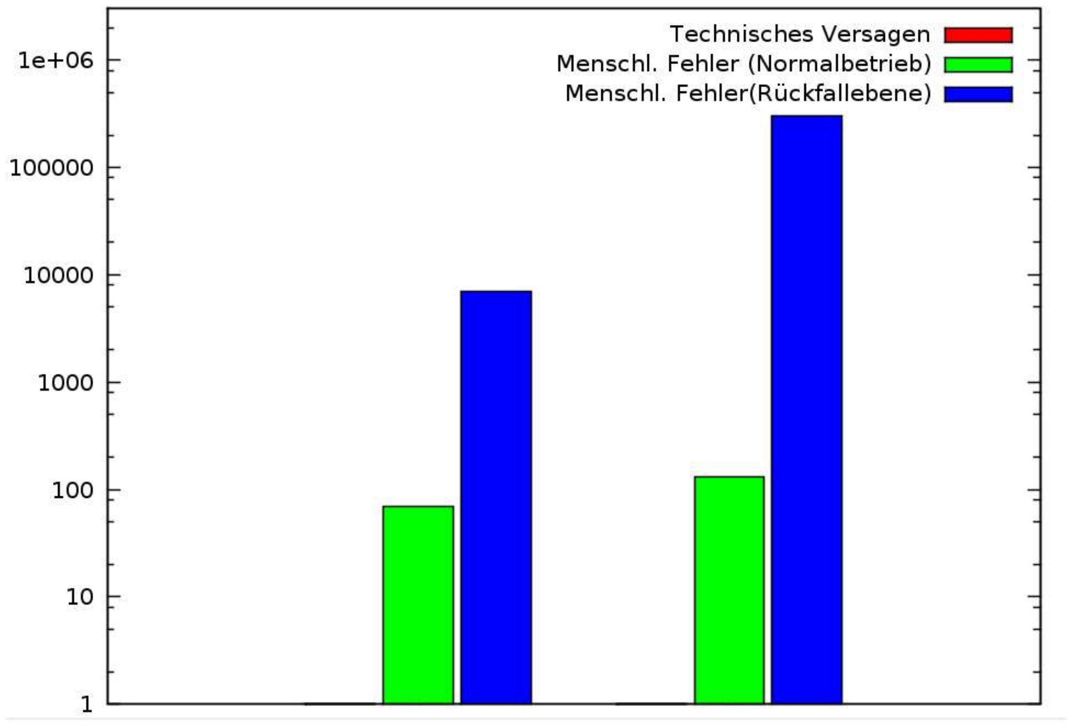
- safety function fails dangerously
- safety function is not available

Security can obviously impact both.

Security in Safety Related Systems



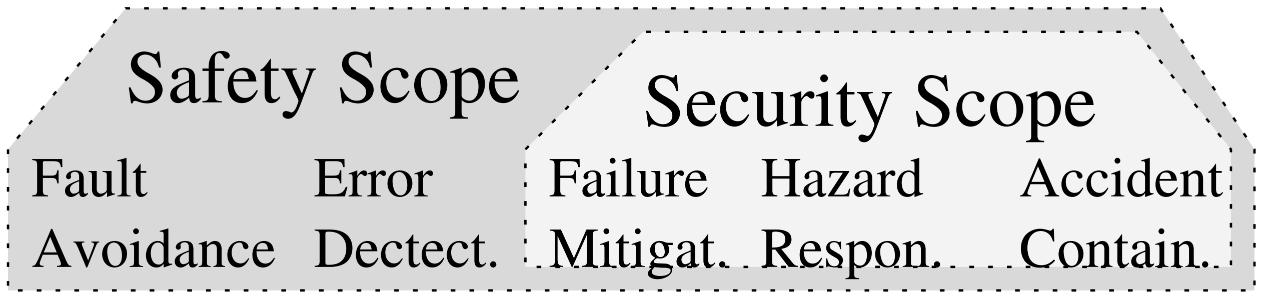
Safety Impact of Service Loss



[Braband: Risikoanalyse in der Eisenbahn Sicherung]

Safety vs. Security

A somewhat simplistic model with respect to focus



Security is mainly concerned with Failure detection and response as well as preventive measures not fault or error detection.

Difference of focus

There are some fundamental differences in approaching corruption of data and code - intentional or not.

- Procedural safety focuses on fault avoidance by procedure
- Evidence based safety focused on fault probability based on history
- Security focuses on fault tolerance and failure detection/response.

Constraints on fault relevance

While security limits it self to responding to failures due to systematic software faults it is not concerned with the class of stochastic faults. Safety is strongly focused on systematic software faults.

The underlying assumption is:

- Security:
  repeatable failures - thus based on systematic faults - are critical failures
- Safety:
  Any fault - including stochastic faults - is potentially dangerous and must be detected.

Rational of constraint

The justification for this security approach is simply that they are concerned with the malicious user - the active component having malicious intentions - and the probability of a ”hacker” trying to enter the pentagon, making it due to a bit flip in the CPU is sufficiently low against the existing software bugs to be ignored.

In safe systems we don’t have any such constraint on the intention of the active component ad there is no distinction between malicious and non malicious interaction (at least not with respect to stochastic faults)

Security in IEC 61508-1 Rev 2

- requires malevolent and unauthorized actions to be considered during hazard and risk analysis and provides informative guidance on the security required for the achievement of functional safety. [IEC 61508-1 Rev 2 1.2 l)]
- 6.2 n) (third item) might need reinterpretation in the security context - "the procedures for preventing unauthorized items from entering service" [IEC 61508-1 Rev 2 6.2 n)]
- The first objective of the requirements of this subclause is to determine the hazards, and hazardous events and hazardous situations relating to of the EUC and the EUC control system (in all modes of operation) for all reasonably foreseeable circumstances, including fault conditions and misuse. [IEC 61508-1 Rev 2 7.4.1.1]

- The hazards, and hazardous events and hazardous situations of the EUC and the EUC control system shall be determined under all reasonably foreseeable circumstances (including fault conditions, and reasonably foreseeable misuse and malevolent or unauthorized action). [IEC 61508-1 Rev 2 7.4.2.3]

Security in IEC 61508-1 Rev 2 - consistency

k) does not cover the precautions that may be necessary to
prevent unauthorized persons damaging, and/or otherwise
adversely affecting, the functional safety of E/E/PE
safety-related systems;
l) requires malevolent and unauthorised actions to be considere
during hazard and risk analysis and provides informative
guidance on the security required for the achievement of
functional safety.
[IEC 61508-1 Ed 2 CD Clause 1.2]

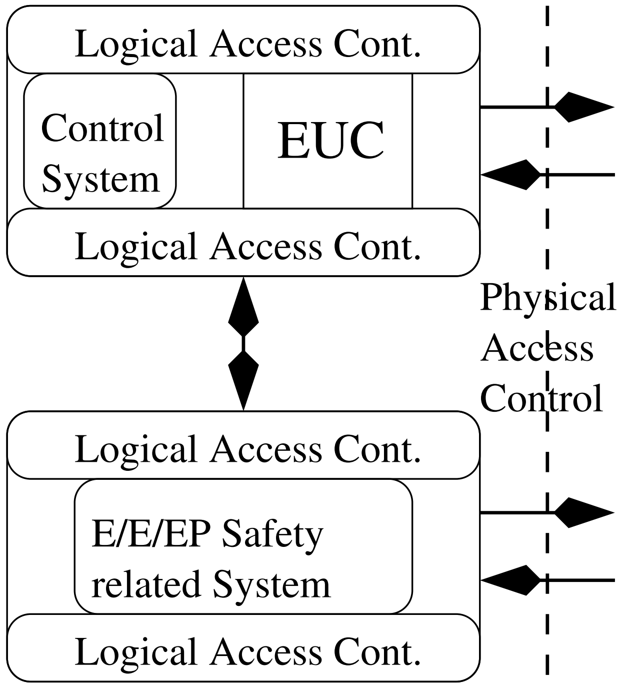
That does sound a bit ignorant of security reality - Imagine a HAZOP where you would need to consider a malicious intent for every transition !

Security is to be considered for

- operation
- maintenance
- repair

"...EUC control systems and E/E/PE safety-related systems operate in one or more security environments that are physically and logically protected and securely separated from the external environment " [IEC 61508-1 Rev 2 Annex B]

Security Model - separate security environments



[IEC 61508–1 CD 2009 Annex B]

Note this is based on 61508 Ed2 CD informative Annex B

High-Level reuse

With respect to high-level properties security and safety does not differ that much - maybe with the exception of the dimension of temporal properties

- guarantee of separation (of different security realms vs. different SILs)
- integrity guarantees (detection of malicious modification vs accidental modification)
- authentication of components (functional assurance - there really is no difference here)
- robustness (DOS vs. babbling idiot problems)

Its more a matter of changing mind-set than changing concepts.

IEC 61508-3 1998

Fundamentally 61508 and derived standards don't dictate a specific integrity level for generic components - but they do put relatively clear requirements on "SIL decomposition":

7.4.2.7 Where the software is to implement both safety and non-safety functions, then all of the software shall be treated as safety-related, unless adequate in dependence between the functions can be demonstrated in the design.

This is in line with security concepts that use domain-models and appropriate verification methods to validate these (i.e. SELinux).

IEC 61508-3 1998

7.4.2.8 Where the software is to implement safety functions of different safety integrity levels, then all of the software shall be treated as belonging to the highest safety integrity level, unless adequate independence between the safety functions of the different safety integrity levels can be shown in the design. The justification for independence shall be documented.

NOTE The software safety integrity level must be at least as high as the safety integrity level of the safety function to which it belongs. However, the safety integrity level of a software component can be lower than the safety integrity level of the the safety function to which the software component belongs, if the component is used in combination with other hardware components such that the safety integrity level of the combination at least equals that of the the safety function.

IEC 61508-3 ED2 CD 2009

NOTE 2 Software elements which execute on a single computer system (processor and memory) can be shown to be independent of each other by means of a number of different methods. Independence must be demonstrated both in the spatial domain (the data used by a high integrity element must not be changed by a lower integrity element) and temporal (the lower integrity elements must not cause the higher integrity elements to function incorrectly by taking too high a share of the available processor execution time). Techniques for achieving and demonstrating spatial independence include

Proof of Separability

John Rushbbys criteria (very informally) - given two domains RED and BLACK then separation means from REDs perspective:

- for every I/O operations on behalf of RED on the concrete machine there is one equivalent abstract operation in the RED domain
- Every RED operation does not impact the perceivable state of the concrete machine for BLACK
- state change in the RED regime caused by RED I/O activity must depend only on the activity itself and the previous state of RED
- BLACK I/O devices cannot change the state of the RED regime
- if RED revisits the same state then the output must be the same
- if RED revisits the same state then the next state must be the same

Some details 7.4.2.8

This is the list of possible technologies listed in 61508 Ed 2 CD as extension in the note2 to 7.4.2.8

- Use of **hardware memory protection** between elements of differing safety integrity levels.
- Where data has to be passed from a higher integrity to a lower integrity element between , use of unidirectional interfaces such as messages or pipes instead of shared memory (data passed from a lower to a higher integrity element should not be permitted unless the higher integrity element can independently verify that **the data is of sufficient integrity**).
- Any data resident on permanent storage devices such as magnetic discs must be taken into account in addition to transient data in

random access memory, for example by means of **file access protection**.

- Use of rigorous design and source code analysis to demonstrate that **no explicit or implicit memory references are made** from the lower to the higher integrity software elements.
- Strict priority based scheduling implemented by a real-time executive with **a means of avoiding priority inversion**.
- **Time fences** which will terminate the execution of a lower integrity element if it over-runs its allotted execution time or deadline.

7.4.2.8 continued

IEC 61508-3 Ed 7.4.2.8 Note 2 Continued

- If an operating system, real-time executive, memory management or timer management software is to be used to provide spatial or temporal independence, or both, then such software must be of the highest safety integrity level of any function.
- Software elements of different criticality monitored by logical and temporal program flow monitoring as per IEC 61508-2 Table A.11.
- Techniques for ensuring temporal independence include deterministic scheduling methods supported by worst case execution time analysis of each element.

Show-stoppers ? still some work to do...

Standards

- ”proof of separability” <–> ARINC 653 (AUTOSAR with limitations)
- EAL Level 5 <–> DO 178-B - highly overlapping
- Security splits requirements into
  - Security Functional Requirements (IEC 15408-2)
  - Security Assurance Requirements (IEC 15408-3

  might be good to replace ”...shall include, commensurate with the required SIL ,...” ”In accordance with the required SIL..” with clearer Assurance requirements.

IEC 15408-2/3 can serve as model for developing clearer functional safety standards without reverting to the check-list mentality of Mu 8004.

Where can security methods help ?

Security methods are not the answer to all safety issues.

- Generic functionality (those that have no application "knowledge" i.e. memcopy, write, malloc...)
- Generic components connecting safety related components
- isolation of safety related components from generic components and each other

The essential thing is that the security methods don't target a specific application or context but protection of generic low-level functions - broadly speaking they all attempt to re-enforce isolation even in the presence of systematic software faults.

Integrity

Basic Integrity related Attack Categories of interest for safety.

- information leakage - environment diversification
- code injection attacks - ISR instruction set randomization
- control flow hijacking - stochastic control-flow diversification
- memory corruption attacks - ASR address space randomization
- File level attacks - IDS/Access control

The security related mechanisms are far more mature than what is currently in use in the safety domain.

Methods of interest

- Address space randomization
- Stack/heap randomization
- inherent randomness / inherent diversity
- Binary Translators (BTs) - (not yet covered)
- Instruction set randomization
- Isolation enhancements
- Multilevel Monitoring (Kenrel,LSM,VFS...)

Note that some of these methods will draw some of the common safety strategies into question (i.e. some forms of replication and recovery).

memory corruption attacks

- They are the simplest to detect
- They are the simplest to implement
- the simplest to mitigate
- memory, notably stacks, are/were quite predictable

GNU/Linux is very much concerned with security - thus current Linux kernels randomize address spaces by default.

ISR

Why ISR will not directly help safety - Fault class: dormant/permanent CPU fault

Instruction set randomization is assuming an attacker that utilizes structural knowledge of the code to inject code - but the safety concern is random code injection respectively CPU fault related injection (which from the code perspective is random)

what studies do show though that such random injections have a low chance of not crashing the application, and the probability of those surviving doing anything useful is also low - notably on complex hardware with complex software!

ISR from a safety perspective could though impact the target of the unintentional jump -> A != B -> detectable.

stochastic control-flow diversification

Note we are ignoring network hijacking - thats a separate bowl of soup.

- reducing predictability of control flow
- increasing variability and perceived complexity in systems
- structure theorem based automated transformation

This can constitute an additional defense against CCF - a kind of fault propagation diversification. More at:
http://www.cert.org/archive/html/stochastic-divers.html

Address Space Randomization

ASR is one of the best studied security methods

- text,data,bss address base randomization
- heap randomization: padding,alignment,random over-allocation
- stack randomization: layout, padding, object order

Address spaces are randomized by default in Linux 2.6, GCC has the ability to add stack randomization.

Esoteric concepts - inherent randomness

- Utilizing growing complexity
- inherent Diversity
- Transition to statistical approach

Modern super-scalar CPUs are non-deterministic beasts - instead of fighting the complexity, it could be utilized !

MLS: Isolation Issues

Security enhancements in GNU/Linux primarily target extended Isolation capabilities:

- Virtualization (KVM,lgues,UML,etc.)
- Kernel Credentials/Capabilities
- Linux Security Modules (SELinux, APARMOR)
- OS-level partitioning: cgroups, cpu-pinning
- Access Control Lists/Extended Attributes

these are useful to cover the growing security demands of safety related systems as well as improve the fault containment capabilities of GNU/Linux.

Conclusions

Functional safety needs some fundamental changes:

- Safety by secrecy does not work - peer review by the safety community is mandatory to achieve adequate safety in systems of ever growing complexity
- A clearer development of assurance requirements is needed - security practice might be a reusable model (IEC 15408-3)
- where security and safety overlap - reuse of heavily scrutinized methods rather than developing from scratch seems prudent.
- Due to the openness applied to security mechanisms ample reviewed field data is available - which is lacking in most dedicated safety related components

Closing Note

Open-Source is not the answer to all functional safety questions but there is a lot to learn from the model notably with respect to how security is treated and peer review policies

safety by secrecy - is one of the most critical systematic faults of the safety community.

http://www.osadl.org/Safety-Critical-Linux.safety-critical-linux.0.html