Security Analysis of Safety Patterns

CHRISTOPHER PRESCHERN, Institute for Technical Informatics, Graz University of Technology
NERMIN KAJTAZOVIC, Institute for Technical Informatics, Graz University of Technology
CHRISTIAN KREINER, Institute for Technical Informatics, Graz University of Technology

Architectural safety patterns provide knowledge about large scale design decisions for safety-critical systems. Safety-critical systems are nowadays increasingly subject to attacks due to their increased connectivity to the Internet. Therefore, we extend existing architectural safety patterns to include security considerations. We apply a STRIDE approach on the safety patterns to obtain relevant threats for each pattern and we structure these threats in a Goal Structuring Notation diagram. We present a catalog of security enhanced safety patterns and we apply one of the patterns to a case study to show how the security-enhanced safety patterns can help for security reasoning.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architecture—*Patterns*; K.4.1 [**Public Policy Issues**] Human Safety; K.6.5 [**Management of computing and information systems**] Security and Protection

ACM Reference Format:

Preschern, C. 2013. Security Analysis of Safety Patterns. Proceedings of PLoP2013. 38 pages.

1. INTRODUCTION

Security concerns are still not sufficiently considered when designing safety-critical systems although they become more relevant due to increasingly interconnected systems. To provide safety engineers with guidelines how to design good systems, safety patterns can be used which describe the safety-related consequences of taking a specific design decision. However, none of the safety patterns in literature extensively cope with the effects of the pattern application on system security.

In this paper we evaluate existing safety patterns regarding their effect on the overall system security. We structurally analyze safety patterns by using the STRIDE approach which is well known in the security domain. This gives us a list of threats for the design patterns which we divide in categories depending on how critical they are for the system's safety. We then present highly critical threats for each pattern in a Goal Structuring Notation diagram, which allows one to easily see which parts of the system are important to protect against attacks. The resulting security enhanced patterns provide a basis for safety engineers to analyze and enhance the security of their systems. We show the application of a pattern and its Goal Structuring Notation diagram in a case study from the substation automation domain.

This paper is structured as follows: Section 2 gives some basic background on the STRIDE approach and on Goal Structuring Notation. Both are used in Section 3 which describes how the security effects of safety patterns are evaluated. Section 4 shows how to apply the security enhanced safety patterns for reasoning about the security of a case study. Section 5 gives related work on security evaluation for design patterns and Section 6 concludes this work. In the Appendix we present our catalog of the security enhanced safety patterns.

2. BACKGROUND

This section gives a basic introduction to the STRIDE threat modeling approach and to Goal Structuring Notation.

2.1 STRIDE Threat Modeling Approach

In order to build a secure system, it is necessary to first find the relevant threats to the system before finding solutions how to mitigate them. The STRIDE approach is a structured way to find these threats. The STRIDE approach was proposed by Microsoft [Howard and LeBlanc 2003] and is nowadays often used as part of security analysis. STRIDE is an acronym, where the letters stand for the six threat categories which are analyzed (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), Elevation of Priviledge (EoP)).

For threat modeling with STRIDE, first a data flow diagram (DFD) has to be constructed. A DFD shows the interaction between system elements and external elements (e.g. users of the system) by graphically presenting all the data flows (inputs/outputs of elements). All relevant STRIDE threats for each element in the diagram are then listed. The relevant threats for different DFD element types are given in Table I.

Table I. STRIDE mapping to DFD element types

DFD element type	S	T	R	I	D	E
External entity	X		X			
Data flow		X		X	X	
Data store		X	X	X	X	
Process	X	X	X	X	X	X

The resulting list of threats can further be elaborated by excluding threats which are not relevant for the specific system and by implementing countermeasures for relevant threats. When all threats are covered, one has a structured argument for system security. We use Goal Structuring Notation to present such a structured argument.

2.2 Goal Structuring Notation

The Goal Structuring Notation (GSN) was developed by [Kelly and Weaver 2004] and is often used in the safety domain for providing a structured argument for the achievement of specific goals. Recently, a standard for the GSN was published which contains definitions of the notation and which presents approaches how to use GSN to elaborate a specific goal [GSN Working Group 2011]. GSN can also be used to argue for system security like in [Cockram and Lautieri 2007]. Figure 1 explains the GSN concepts which are later on used in this paper.

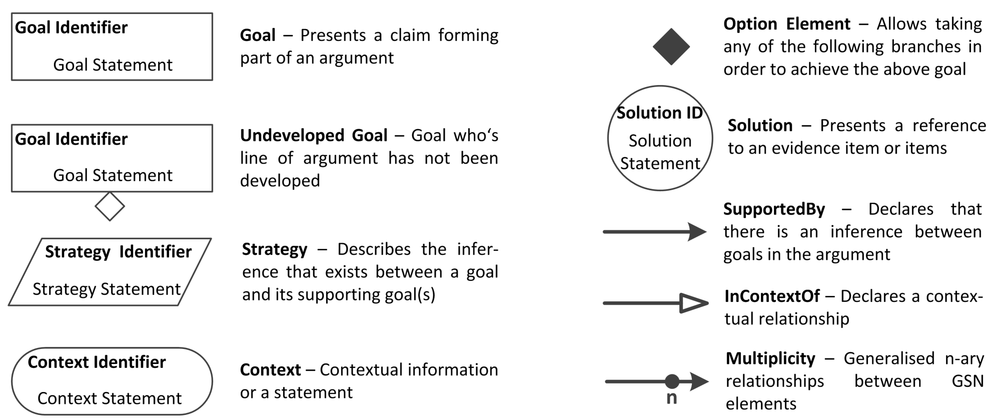


Fig. 1. GSN concepts used in this paper taken from [GSN Working Group 2011]

To show how a GSN goal is achieved, it is linked to an argument (GSN strategies, GSN subgoals) which ends up in the evidence (GSN solution) supporting the claim that the goal is achieved. Figure 2 shows an example for the application of GSN. The main goal in the example is that an attacker cannot obtain some confidential data. In the next step, context elements are added which say that the data is locally stored on a computer and transmitted to another computer. The main goal is split up into the subgoals to protect the stored data and the transmitted data. Protecting transmitted data is achieved by just transmitting the data over a protected TLS channel (GSN strategy). For this TLS channel, we need evidence that it works properly. This evidence (GSN solution element) is that the used implementation is security certified. Protecting stored data is an undeveloped goal which means that the security argument for this subgoal is not yet complete and further arguments have to be included here in order to obtain a complete argument that the overall goal (protecting the confidential data) is achieved. In the example, GSN provides a structured way to show how the rather unspecific goal to protect confidential data is (partially) achieved by specific measures (the TLS channel).

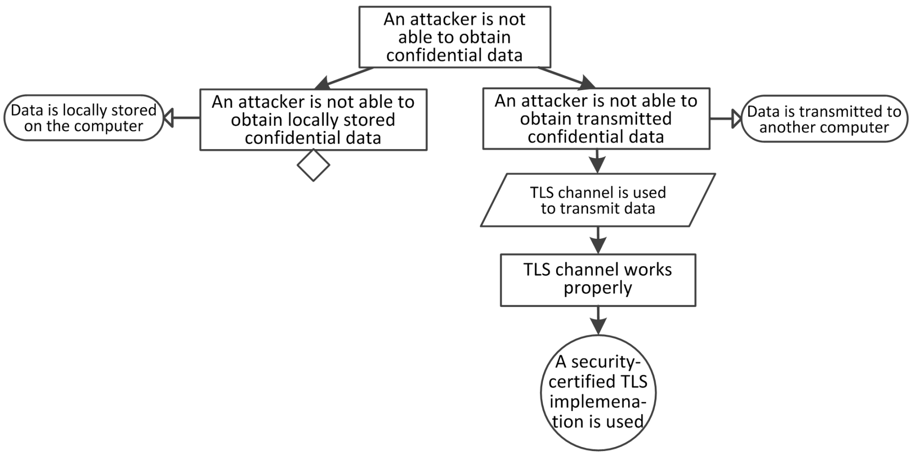


Fig. 2. GSN example showing a security argument

2.3 Alternative Methods for Threat Elaboration

STRIDE is a very generic security analysis and can be used as a starting point to develop security requirements and security countermeasures. In this work we build a GSN diagram to argue for STRIDE threat mitigation; however, instead of GSN several alternatives could be used:

- The Secure Tropos project [Mouratidis and Giorgini 2007] provides a tool to model a system and to analyze its threats with the STRIDE method [Rojas and Mahdy 2011]. Threat mitigation mechanisms can be added to the system model and security reports can be printed with the tool. The reason for using GSN diagrams instead to the Secure Tropos model representation including tooling support is that the GSN notation is well known in the safety domain.
- Microsoft's security development lifecycle suggests to build threat trees for each STRIDE threat and to mitigate each element of such a threat tree. The reason for using GSN diagrams instead of threat trees is that GSN diagrams easily allow to integrate security countermeasures into the notation and to further analyze STRIDE threats for these countermeasures. With the threat tree notation that would be cumbersome.
- Fault Trees can also be used to integrate security threats [Nai-Fovino et al. 2009]. However, The reason for choosing GSN above fault trees is that GSN is already known and applied in the safety and security domain whereas fault trees are usually just used in the safety domain.

3. ENHANCING PATTERNS WITH SECURITY REASONING

In this section we present and apply the approach how to use the STRIDE analysis for safety patterns in order to obtain a GSN argument for the patterns which helps to identify threats and to argue for the security of a system which applies a pattern.

3.1 Getting the Data Flow Diagram

The catalog of safety architecture patterns presented in [Preschern et al. 2013] shows several safety patterns with a consistent notation. Each of the patterns describes how a *Basic System* consisting of hardware or software elements can be modified (e.g. through adding a watchdog, or through replication) in order to increase its safety. Each of the patterns provides a diagram which shows the hardware and software elements of the pattern and their interaction. For the patterns, this diagram contains all the necessary information for the STRIDE analysis and will be used instead of a data flow diagram.

Figure 3 shows such a diagram for the *Basic System* (to which the patterns from [Preschern et al. 2013] can be applied) The *Basic System* gets input data, processes that input data in the primary channel, and produces output data for a safety-critical process.

Fig. 3. Basic system which is the starting point for the safety patterns

3.2 Getting the Threats

By using an adapted STRIDE approach, we analyze the pattern diagrams to list the security threats for each of the patterns.

We just consider two element types for the STRIDE analysis: *Data flows* and *Processing elements*. For both types, we omit the threats Repudiation and Information Disclosure, because they do not directly influence the safety functionality of a system. Furthermore, for the *Processing elements*, we omit the Tampering and Denial of Service threats, because an attacker usually has no access to processing elements which perform safety-critical functionality. Therefore, he needs to elevate his privileges before starting a tampering or DoS attack on a processing element. Our resulting relevant threats for the pattern diagram element types are shown in Table II.

Table II. STRIDE mapping to safety pattern element types

DFD element type	**S**	**T**	**R**	**I**	**D**	**E**
Data flow		X			X	
Processing element	X					X

With this mapping of relevant threats, we go through each element of the pattern diagram to obtain a list of relevant threats for the pattern. For the *Basic System* we get the following list of threats:

- Tampering of Primary Channel input
- DoS against Primary Channel input
- Spoofing of the Primary Channel
- EoP on the Primary Channel
- Tampering of Primary Channel output
- DoS against the Primary Channel output

3.3 Categorizing the Threats

For each pattern we divide the obtained threats into criticality categories which make it easier to quickly see which threats are especially relevant for the pattern. The threats are categorized as:

- Threats to the safety-critical functionality of the system
- Threats which can bring the system into a safe state (e.g. shut it off)
- Threats which do not directly influence the system functionality and leave the system fully functional

To determine which category a threat belongs to, we analyze what would happen if a successful attack related to the threat was applied. If the attack could arbitrarily modify the system's output data, then the threat is **safety-critical**. If the attack could shut the system off, then the threat is classified as one which **leads to a safe state**. If the attack does not influence the system's output, the threat is classified as one where the **system remains fully functional**.

We display the categorized threats in a table which lists them according to their STRIDE type and criticality category. All the threats for the *Basic System* are underlines and printed in green color. All other threats (threats for a safety pattern apart from the *Basic System* threats), are printed in black color. This has the advantage that for the safety patterns in the Appendix, one can easily see to which criticality category the *Basic System's* threats are shifted when applying the pattern or whether the *Basic System's* threats are then even relevant anymore.

For the *Basic System*, after applying the described threat categorization, we obtain the threat table shown in Table III. We can see that all threats are categorized as safety-critical. For example, the *"Tampering of Primary Channel input"* threat is safety-critical, because if someone can maliciously modify the Primary Channel input data, then, in general, it is also possible to modify the system's output data, because the output data calculation of the primary channel depends on the input data. We can also see that all threats underlined and printed in green. This is, because per definition, we print all *Basic System* threats underlined and in green. When looking at the patterns in the Appendix, also threats printed in black are present and the advantage of using different colors in the table can be seen, because one can easily see which basic threats (underlined, green) are shifted into other columns. This gives a quick overview of how the pattern affects the existing threats.

Table III. STRIDE threats relevant for the *Basic System*

	safety-critical	**leads to a safe state**	**system remains fully functional**
S	Spoofing of the Primary Channel	-	-
T	Tampering of Primary Channel input Tampering of Primary Channel output	-	-
R	-	-	-
I	-	-	-
D	DoS against Primary Channel input DoS against Primary Channel output	-	-
E	EoP on Primary Channel	-	-

To highlight the safety-critical threats, we color all components in the pattern's diagram which are related to safety-critical threats in red. For the *Basic System*, this was already done in Figure 3. For the patterns presented in the Appendix, this makes it very easy to get a first impression of which components especially have to be protected. All the patterns in the Appendix contain a table with their categorized threats.

3.4 Constructing the Security GSN

In some cases, threats which are not classified as safety-critical can become part of an attack affecting system safety if they are combined. To also capture thi s information we construct a GSN diagram for each pattern. The top-level GSN goal is *to maintain the safety functionality even in case of an attack*. The subgoals are the prevention of attacks leading to the analyzed safety-critical threats or the prevention of attack combinations[1].

Using GSN diagrams to represent attacks is not the original approach presented by [Howard and LeBlanc 2003] for the STRIDE method. The original approach is to gather attacks and use a tree-like notation (called attack trees) to display how these attacks can be combined to form STRIDE threats. However, attack trees just capture the information how to relate attacks and do not contain information about the countermeasures against these attacks. With GSN it is possible to relate countermeasures (GSN strategies) to the attack which they mitigate (GSN goals). Thus, compared to attack trees, GSN diagrams bring the advantage of establishing a link between the security goals (protect against system threats) and the implemented countermeasures. A similar approach was already suggested by [Moleyar and Miller 2007].

Figure 4 shows the security GSN diagram for the *Basic System* which is rather straightforward, because all its threats are safety-critical. However, if we would construct a GSN for a system similar to the *Basic System* but which additionally has a safe state when it is shut off, we would obtain a slightly different GSN diagram. If the system had a safe state when shut off, the DoS threats would not be safety critical, but they would belong to the second column ("leads to a safe state") in Table III. For the GSN diagram this would mean that the DoS threats would not be part of it, because they cannot lead the system to a critical state (also not if both DoS threats would be combined).

All of the patterns in the Appendix contain a security GSN diagram. These diagrams are more complex than the diagram in Figure 4 and yield additional information regarding the possible threat combinations which are safety-critical. Such a GSN diagram can then be used as a basis for security reasoning for a specific architecture which applies one of the safety patterns. The GSNs of the patterns contain undeveloped goals, because the implementation details for a specific architecture applying one of the patterns are not yet known. These undeveloped goals have to be developed (by adding architecture-specific claims and proves that support the goal) to obtain a complete security argumentation.

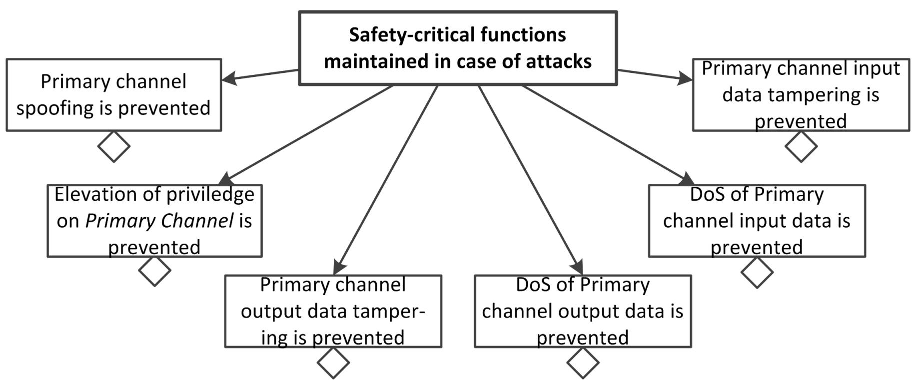


Fig. 4. Security GSN diagram for the basic system

[1] To model a combination of attacks, the subgoals would be related with a GSN option element to the main goal - the HETEROGENOUS DUPLEX PATTERN is an example which contains a combined attack and therefore makes use of the GSN option element

4. APPLYING THE SECURITY ENHANCED SAFETY PATTERNS TO A CASE STUDY

In this section we apply one of the safety patterns from the Appendix to a case study. With the security analysis of the safety pattern, we construct a complete security argument for the system architecture.

4.1 System description

In our case study we apply the HETEROGENOUS DUPLEX PATTERN to an electrical substation automation device. Substations handle functions like voltage protection and conversion between different voltage distribution networks. The substation automation device in our case study is a safety-critical component which handles over-voltage protection. Based on measured current and voltage input values, the device has to decide if a power distribution network should be cut off in order to protect other devices from over-voltage. The over-voltage protection device obtains its sensor inputs from an IEC61850 merging unit, which is a sensor unit distributing sensor values via Ethernet. Based on this sensor data, the system has to control actuators which are hardwired to the device. The system is connected to the local substation Ethernet network to enable firmware updates.

Figure 5 gives an overview of the system architecture after applying the HETEROGENOUS DUPLEX PATTERN (more details about the pattern are given in the Appendix on page 15). The substation automation device has two CPUs where each CPU input is supplied with its own set of sensor data. To compute the actuator output value, the CPUs run diverse software versions. This means the software versions have the same functionality, but different implementations. Each CPU runs a diagnostic test and periodically sends the results of the test to an FPGA which checks the diagnostic results and switches the actuator output to the backup CPU output if the diagnostic test of the primary CPU fails. An external connection to both CPUs can be established via the local Ethernet to install firmware updates on the CPUs.

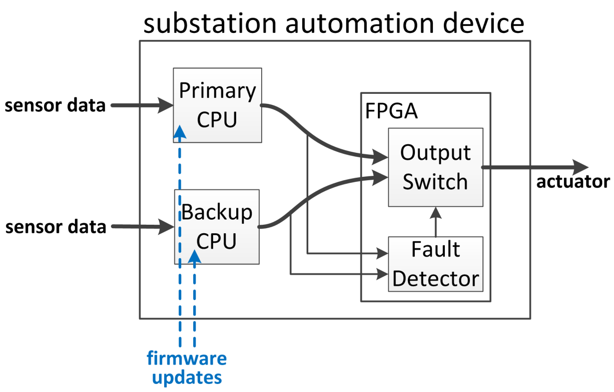


Fig. 5. Substation automation device architecture

The architecture is very similar to the basic HETEROGENOUS DUPLEX PATTERN which is described in the Appendix. The only differences are that the architecture has an additional connection to the CPUs for firmware updates and that the fault detector and the output switch are realized on a single hardware component.

4.2 Adapting the security GSN from the pattern

The HETEROGENOUS DUPLEX PATTERN includes a security GSN diagram which captures the aim to mitigate safety-relevant threats for this pattern as subgoals. These subgoals are undeveloped GSN goals (because the GSN diagram of the pattern does not yet include information how these subgoals are achieved). We now want to develop the subgoals in order to obtain a complete security argument for our architecture. We go through every undeveloped goal and check whether the threat is actually a threat for the specific architecture. If it is not, we add the information why it is not a relevant threat to the GSN notation. If it is a relevant threat, we suggest mitigation strategies. Figure 6 shows the resulting security GSN diagram for the substation automation device architecture. Black elements with solid lines are taken from the GSN diagram of the HETEROGENOUS DUPLEX PATTERN and green, dashed elements are added for the specific architecture.

The completed GSN diagram in Figure 6 shows us that some of the threats to the system (e.g. the *"DoS of Fault Detector is prevented"* GSN goal element) are irrelevant. However, we do not eliminate these elements from the diagram, but add GSN elements which argue why these threats are sufficiently handled by the architecture itself (e.g. *"An attacker has no physical access to the Fault Detector"* GSN context element and *"The Fault Detector is hardwired to the CPU diagnosis output"* GSN solution element).

Some other threats are not irrelevant but require countermeasures. For example, to mitigate the EoP threats to the switch, to the fault detector, and to the CPUs, the countermeasure to thoroughly test these units (GSN strategy elements) and to provide the test results (GSN solution elements) is applied. Additionally, for the CPUs, the countermeasure to check the integrity of firmware updates is applied to handle the threat of achieving EoP on the CPU by using a malicious firmware update. Another set of threats which have to be mitigated with appropriate countermeasures, are threats to the merging unit. These threats are countered by putting the merging unit into a separate Ethernet network to which an attacker does not have access.

4.3 Benefits of the Security GSN diagram

The main benefit of the GSN diagram is that with the application of a safety architecture pattern, we get a structured representation of relevant security threats. This allows us on the one hand to argue for the overall system security and on the other hand points to weaknesses of the architecture. By not deleting irrelevant threats but adding information to the GSN diagram why these threats are irrelevant, we obtain a security argument for the architecture which is complete regarding its safety-relevant STRIDE threats.

5. RELATED WORK

This section covers related work on the security evaluation of safety-critical systems and on the security evaluation of design patterns.

[Hansen 2009] presents a security analysis of a safety-critical automation device which highlights attacks compromising the system safety. [Johnson and Yepez 2011a] and [Johnson and Yepez 2011b] presents a combined security and safety risk assessment methodology where security and safety arguments are shown in a GSN diagram. Security threats are analyzed for a case study and the threats are included in an existing safety GSN to obtain a unified assurance case for safety and security. [Nai-Fovino et al. 2009] present a method to integrate security reasoning into fault trees. They discuss how to analyze the risk of security aspects in order to integrate their probabilities consistently into the fault tree notation. A similar apprach is taken by [Ugljesa and Wacker 2011] to integrate security considerations into the error probability calculation of a 2oo4 architecture[2]. [Yampolskiy et al. 2012] present an extension of data flow diagrams which allows analyzing an architecture for STRIDE attacks as well as for safety.

[Yautsiukhin and Scandariato 2008] conduct a STRIDE analysis for a case study and discuss how well several patterns can counter the threats. They use a risk assessment method to rate the threat severity and they assign a value to each pattern describing how well the pattern copes with different threats. With this method the security of different patterns for a system can be quantitatively compared. A similar approach is taken in [Halkidis et al. 2006b], [Halkidis et al. 2006a], and [Halkidis et al. 2008]. They evaluate the effectiveness of web security patterns against STRIDE attacks by experiments. With these results they suggest patterns for a web system by first conducting a STRIDE analysis for the concrete system and then suggesting the patterns which mitigate the STRIDE attacks best. This work is also done for security patterns in general in [Halkidis et al. 2004], where a mapping between

[2]The 2oo4 architecture is a special version of the M-OUT-OF-N PATTERN which is explained on page 19.

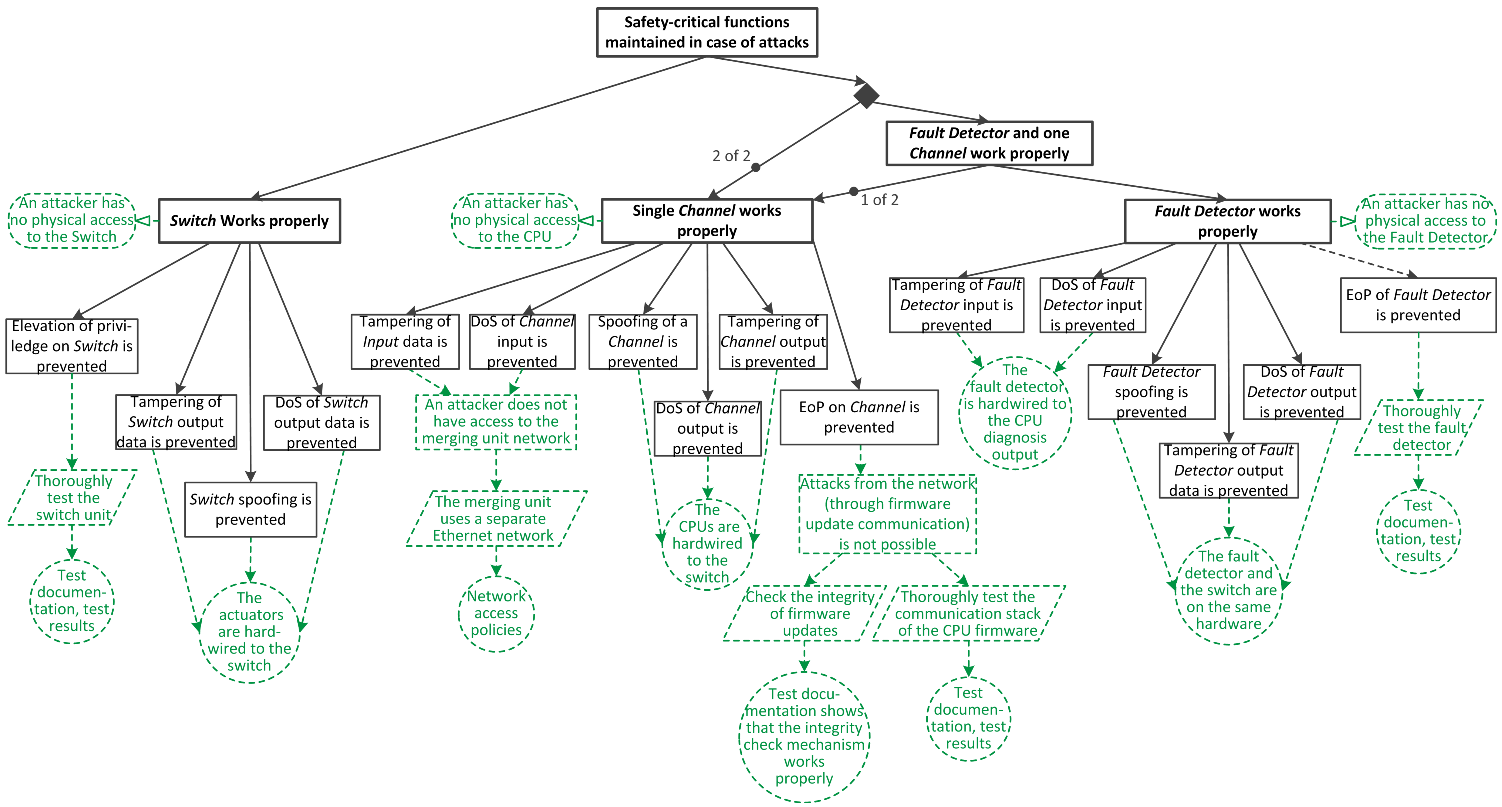


Fig. 6. Substation automation device security argument (based on the GSN of the HETEROGENOUS DUPLEX PATTERN)

several security patterns and their effectiveness for STRIDE attacks is presented. In [Schaad and Borozdin 2012] and [Schaad and Garaga 2012] a tool is presented which reports threats for an architecture by automatically applying the STRIDE analysis to an architecture model. As in our approach, the STRIDE analysis is adapted to just include the threats relevant for the specific architecture element types. [Hamid et al. 2010] take another approach with the TERESA project, by applying a model-based approach to integrate design patterns in order to argue about the safety and security of a system. The tool-based process of how to apply the design patterns is described in [Hamid et al. 2013].

6. CONCLUSION

In this paper we added a GSN diagram describing security threats to safety architecture patterns and we discussed the application of these security enhanced safety patterns to a case study.

The safety patterns described in the Appendix all provide a data flow diagram. Therefore, it is easy to analyze the security of the safety patterns by using the STRIDE approach. All of the described safety patterns enhance the same basic system which makes it possible to compare the security attributes of the different patterns.

The main benefits of the security GSN diagram are that it provides a structured argument for the security of a safety system and that it indicates security flaws of the design. Another important benefit of the security enhanced patterns is that safety experts who use these patterns are confronted with the STRIDE approach. This increases the awareness of security threats in the safety domain which is in our opinion not sufficiently addressed so far.

ACKNOWLEDGMENTS

We would like to thank our shepherd Robert Hanmer who significantly helped to improve this paper by providing us with very helpful feedback on the paper structure and contents as well as on its style.

REFERENCES

ARMOUSH, A. 2010. Design patterns for safety-critical embedded systems. Ph.D. thesis, RWTH Aachen University.

COCKRAM, T. J. AND LAUTIERI, S. R. 2007. Combining Security and Safety Principle in Practice. In *2nd Institution of Engineering and Technology International Conference on System Safety*. IEEE, 159–164.

DOUGLASS, B. P. 2002. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems.* Pearson.

GRUNSKE, L. 2003. Transformational Patterns for the Improvement of Safety Properties in Architectural Specification. In *Proceedings of The Second Nordic Conference on Pattern Languages of Programs (VikingPLoP)*.

GSN WORKING GROUP. 2011. GSN Community Standard Version 1. http://www.goalstructuringnotation.info/.

HALKIDIS, S., CHATZIGEORGIOU, A., AND STEPHANIDES, G. 2004. A qualitative evaluation of security patterns. In *6th International Conference on Information and Communications Security*. Springer, 132–144.

HALKIDIS, S., TSANTALIS, N., CHATZIGEORGIOU, A., AND STEPHANIDES, G. 2008. Architectural Risk Analysis of Software Systems Based on Security Patterns. *IEEE Transactions on Dependable and Secure Computing 5,* 3, 129–142.

HALKIDIS, S. T., CHATZIGEORGIOU, A., AND STEPHANIDES, G. 2006a. A qualitative analysis of software security patterns. *Computers & Security 25,* 5, 379–392.

HALKIDIS, S. T., CHATZIGEORGIOU, A., AND STEPHANIDES, G. 2006b. Quantitative Evaluation of Systems with Security Patterns Using a Fuzzy Approach. In *Proceedings of the 2006 international conference on On the Move to Meaningful Internet Systems: AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET*. Springer, 554–564.

HAMID, B., DESNOS, N., GREPET, C., AND JOUVRAY, C. 2010. Model-based security and dependability patterns in RCES - the TERESA Approach. In *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems - S&D4RCES '10*. ACM Press, New York, New York, USA.

HAMID, B., GEISEL, J., ZIANI, A., BRUEL, J.-M., AND PEREZ, J. 2013. Model-Driven Engineering for Trusted Embedded Systems Based on Security and Dependability Patterns. In *16th International SDL Forum*. Springer, 72–90.

HANMER, R. S. 2007. *Patterns for Fault Tolerant Software.* Wiley.

HANSEN, K. 2009. Security attack analysis of safety systems. *IEEE Conference on Emerging Technologies & Factory Automation*, 1–4.

HOWARD, M. AND LEBLANC, D. 2003. *Writing Secure Code.* Microsoft Press.

JOHNSON, C. W. AND YEPEZ, A. A. 2011a. Cyber Security Threats to Safety-Critical Space-Based Infrastructures. In *Proceedings of the Fifth Conference of the International Association for the Advancement of Space Safety*. Number 1.

JOHNSON, C. W. AND YEPEZ, A. A. 2011b. Mapping the Impact of Security Threats on Safety-Critical Global Navigation Satellite Systems. In *Proceedings of the 29th International Systems Safety Society*. Number 1. International Systems Safety Society.

KELLY, T. AND WEAVER, R. 2004. The Goal Structuring Notation Ű A Safety Argument Notation. In *Proceedings of the Dependable Systems and Networks Conference*.

MOLEYAR, K. AND MILLER, A. 2007. Formalizing attack trees for a SCADA system. In *International Conference on Critical Infrastructure Protection*. IFIP.

MOURATIDIS, H. AND GIORGINI, P. 2007. Secure Tropos: A Security-Oriented Extension of the Tropos Methodology. *International Journal of Software Engineering and Knowledge Engineering 17,* 2, 23–36.

NAI-FOVINO, I., MASERA, M., AND DE-CIAN, A. 2009. Integrating cyber attacks within fault trees. *Reliability Engineering & System Safety 94,* 9, 1394–1402.

PRESCHERN, C., KAJTAZOVIC, N., AND KREINER, C. 2013. System of safety-critical embedded Architecture Patterns. In *EuroPLoP*.

ROJAS, D. M. AND MAHDY, A. M. 2011. Integrating Threat Modeling in Secure Agent-Oriented Software Development. *International Journal of Software Engineering 2,* 2, 23–36.

SCHAAD, A. AND BOROZDIN, M. 2012. TAM2: Automated Threat Analysis. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 1103–1108.

SCHAAD, A. AND GARAGA, A. 2012. Automating architectural security analysis. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*. ACM, 131–132.

UGLJESA, E. AND WACKER, H.-D. 2011. Modeling Security Aspects in Safety Environment. In *7th International Conference on Electrical and Electronics Engineering*. 46–50.

YAMPOLSKIY, M., HORVATH, P., KOUTSOUKOS, X. D., XUE, Y., AND SZTIPANOVITS, J. 2012. Systematic analysis of cyber-attacks on CPS-evaluating applicability of DFD-based approach. In *5th International Symposium on Resilient Control Systems*. IEEE, 55–62.

YAUTSIUKHIN, A. AND SCANDARIATO, R. 2008. Towards a quantitative assessment of security in software architectures. In *13th Nordic Workshop on Secure IT Systems (NordSec)*.

A. SECURITY ENHANCED SAFETY PATTERNS

This section presents the safety patterns from [Preschern et al. 2013] with the security notation described in the previous section. We extend all the safety patterns which we already related to a pattern system in a previeous work [Preschern et al. 2013]. Figure 7 shows this pattern system.

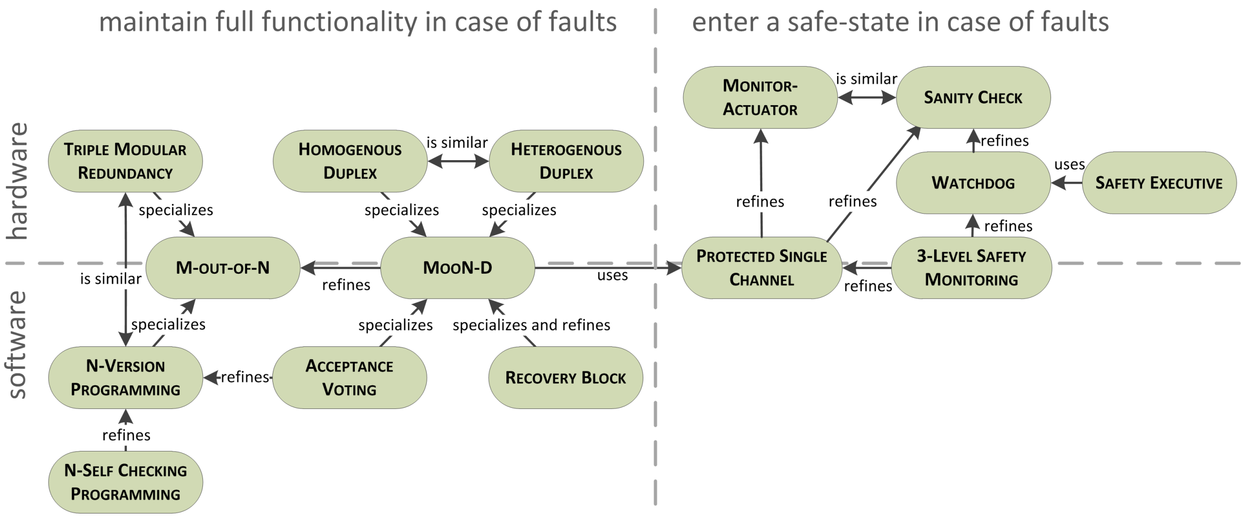
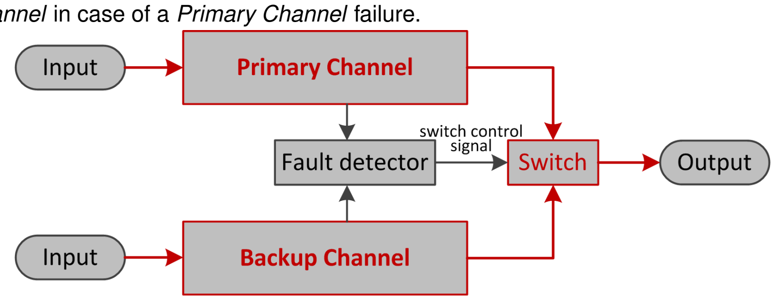
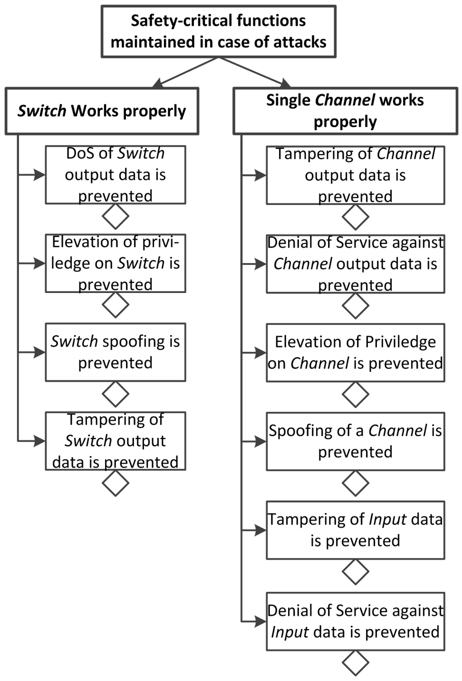
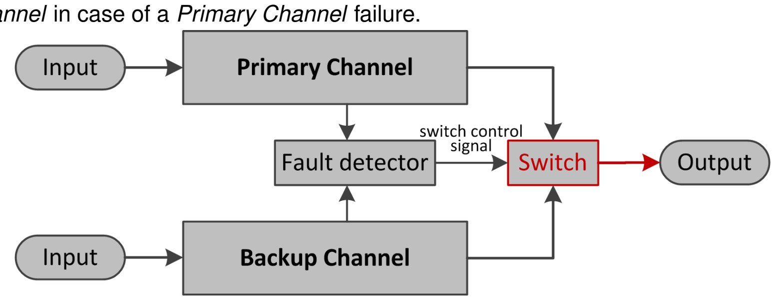


Fig. 7. Safety architecture pattern system from [Preschern et al. 2013]

<table>
<tr><td>Pattern Name</td><td>HOMOGENOUS DUPLEX PATTERN</td><td>Pattern Type</td><td>hardware, fail-over</td></tr>
<tr><td>Also Known As</td><td colspan="3">Homogeneous Redundancy Pattern, Standby-Spare Pattern, Dynamic Redundancy Pattern, Two-Channel Redundancy Pattern, 1oo2D Pattern</td></tr>
<tr><td>Context</td><td colspan="3">A safety-critical application without a fail-safe state has a high random error rate and a low systematic error rate.</td></tr>
<tr><td>Problem</td><td colspan="3">How to design a system which continues operating even in the presence of a fault in one of the system components</td></tr>
<tr><td>Forces</td><td colspan="3">- the system cannot shut down because it has no safe state
- development costs should not increase
- the safety standard requires high fault coverage for single-point of failure components
- high availability requires hardware platforms to be maintained at the runtime</td></tr>
<tr><td>Solution</td><td colspan="3">The system consists of a Primary Channel (active) and a Secondary Channel (backup) which are two identical hardware modules. A Fault detector monitors the channels and controls a Switch to select the Backup Channel in case of a Primary Channel failure.

</td></tr>
<tr><td>Security GSN</td><td colspan="3">The switch is a single point of failure of this pattern and has to be well protected against security flaws. The two replicated channels are identical and therefore it is very likely that an attacker who can compromise one channel, can also compromise the other channel without a lot of effort.
<table>
<tr><th></th><th>safety-critical</th><th>leads to a safe state</th><th>system remains fully functional</th></tr>
<tr><td>S</td><td>Spoofing of the Switch
Spoofing single Channel</td><td>-</td><td>Spoofing of Fault Detector</td></tr>
<tr><td>T</td><td>Tampering of Switch output data
Tampering of single Channel input data
Tampering of single Channel output data</td><td>-</td><td>Tampering of Fault Detector input
Tampering of Fault Detector output</td></tr>
<tr><td>R</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>I</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>D</td><td>DoS of Switch output data
DoS of single Channel input
DoS of single Channel output</td><td>-</td><td>DoS of Fault Detector input
DoS of Fault Detector output</td></tr>
<tr><td>E</td><td>EoP on Switch
EoP on single Channel</td><td>-</td><td>EoP on Fault detector</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td colspan="2">
</td></tr>
<tr><td rowspan="4">Consequences</td><td colspan="2">Systematic and random faults in a single channel are detected and masked.
System reliability strongly depends on the fault coverage of the fault detection unit and on the proper functionality of the switch.</td></tr>
<tr><td colspan="2">Affected Attributes</td></tr>
<tr><td>Positively</td><td>Negatively</td></tr>
<tr><td>Safety: Random Errors in a single channel are handled
Availability: The full system functionality is still available in case of a single random fault
Maintenance: Hardware channels can be maintained at runtime</td><td>Double hardware costs for system replication</td></tr>
<tr><td>Credits</td><td colspan="2">[Douglass 2002] introduces the pattern. [Grunske 2003] presents a more general version of this pattern and [Armoush 2010] adds detailed information about quality attribute related consequences.</td></tr>
</table>

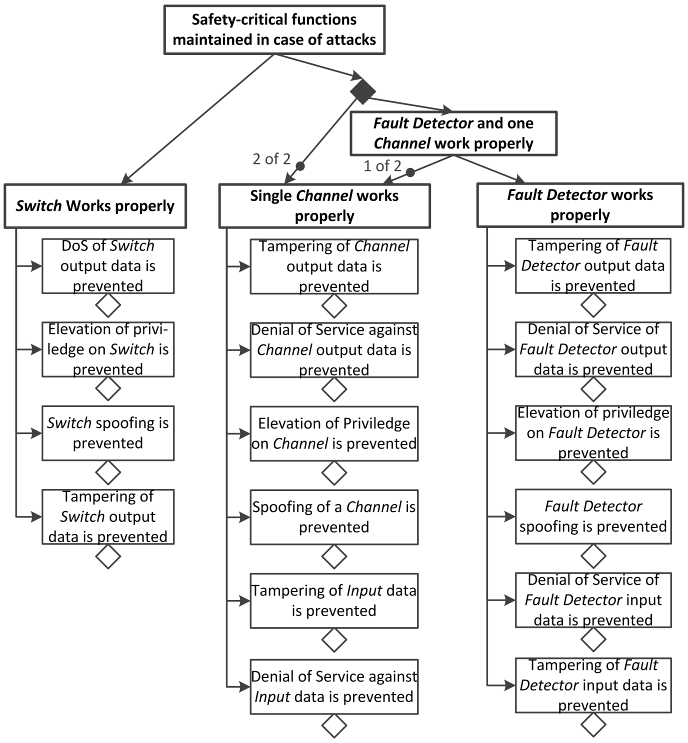
Pattern Name	HETEROGENOUS DUPLEX PATTERN	**Pattern Type**	hardware, fail-over
Also Known As	Heterogenous Redundancy Pattern, Diverse Redundancy Pattern, 1oo2D Pattern		
Context	A safety-critical application without a fail-safe state has a high random and systematic error rate.		
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components		
Forces	- the system cannot shut down because it has no safe state - high safety certification levels require handling of systematic faults - the safety standard requires high fault coverage for single-point of failure components - high availability requires hardware platforms to be maintained at the runtime		
Solution	The system consists of a *Primary Channel* (active) and a *Secondary Channel* (backup) which are two diverse hardware modules. A *Fault detector* monitors the channels and controls a *Switch* to select the *Backup Channel* in case of a *Primary Channel* failure.		



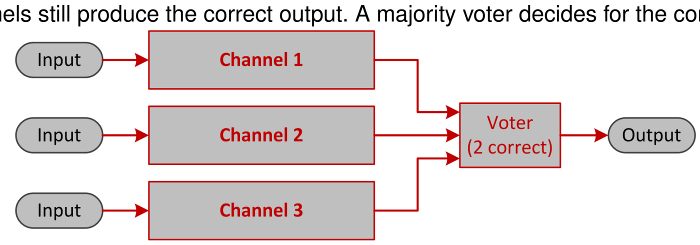
Security GSN

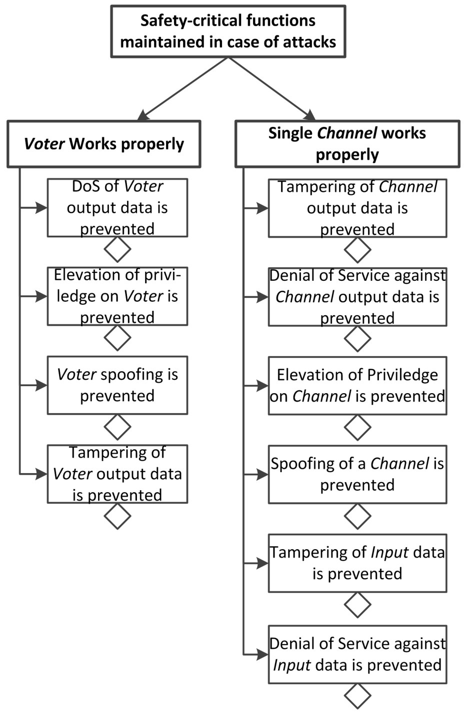
The switch is a single point of failure of this pattern and has to be well protected against security flaws.

	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of the *Switch*	-	Spoofing of a single *Channel* Spoofing of the *Fault Detector*
T	Tampering of *Switch* output data	-	Tampering of single *Channel* input data Tampering of single *Channel* output data Tampering of *Fault Detector* input data Tampering of *Fault Detector* output data
R	-	-	-
I	-	-	-
D	DoS of *Switch* output data	-	DoS of single *Channel* input data DoS of single *Channel* output DoS of *Fault Detector* output data DoS of *Fault Detector* input data
E	EoP on *Switch*	-	EoP on single *Channel* EoP on *Fault Detector*

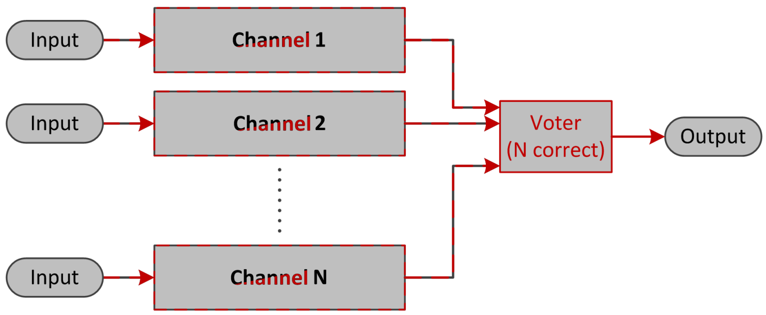


Consequences	Systematic and random faults in a single channel are detected and masked. System reliability strongly depends on the fault coverage of the fault detection unit, on the proper functionality of the switch, and on the level of diversity between the two channels	
	Affected Attributes	
	Positively	**Negatively**
	Safety: Random and systematic faults in a single channel are detected and handled *Availability:* The full system functionality is still available in case of a single random or systematic fault *Maintenance:* Hardware channels can be maintained at runtime	Double hardware costs for system replication, Double development costs due to diverse channels, Modifying the functionality of a channel requires double effort
Credits	[Douglass 2002] introduces the pattern. [Grunske 2003] presents a more general version of this pattern and [Armoush 2010] adds detailed information about quality attribute related consequences.	

<table>
<tr><td>Pattern Name</td><td>TRIPLE MODULAR REDUNDANCY PATTERN</td><td>Pattern Type</td><td>hardware, fail-over</td></tr>
<tr><td>Also Known As</td><td colspan="3">2oo3 Pattern, Homogeneous Triplex Pattern</td></tr>
<tr><td>Context</td><td colspan="3">A safety-critical application without a fail-safe state, a high random error and a low systematic error rate.</td></tr>
<tr><td>Problem</td><td colspan="3">How to design a system which continues operating even in the presence of a fault in one of the system components.</td></tr>
<tr><td>Forces</td><td colspan="3">- the system cannot shut down because it has no safe state
- the safety standard requires high fault coverage for single-point of failure components
- high availability requires hardware platforms to be maintained at the runtime</td></tr>
<tr><td>Solution</td><td colspan="3">Three identical hardware channels operate in parallel. If a single fault occurs in one channel then the other two channels still produce the correct output. A majority voter decides for the correct result.

</td></tr>
<tr><td>Security GSN</td><td colspan="3">The voter is a single point of failure of this pattern and has to be well protected against security flaws. The three replicated channels are identical and therefore it is very likely that an attacker who can compromise one channel, can also compromise the other channels without a lot of effort.
<table>
<tr><th></th><th>safety-critical</th><th>leads to a safe state</th><th>system remains fully functional</th></tr>
<tr><td>S</td><td>Spoofing of Voter
Spoofing of Channel</td><td>-</td><td>-</td></tr>
<tr><td>T</td><td>Tampering of Voter output
Tampering of Channel input
Tampering of Channel output</td><td>-</td><td>-</td></tr>
<tr><td>R</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>I</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>D</td><td>DoS of Voter output
DoS of Channel input
DoS of Channel output</td><td>-</td><td>-</td></tr>
<tr><td>E</td><td>EoP on Voter
EoP on Channel</td><td>-</td><td>-</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td colspan="2">
</td></tr>
<tr><td rowspan="4">Consequences</td><td colspan="2">This pattern does not identify the type or the reason of the fault; it just determines the module that contains a fault without correcting the fault itself. The voter has to be very reliable.</td></tr>
<tr><td colspan="2">Affected Attributes</td></tr>
<tr><td>Positively</td><td>Negatively</td></tr>
<tr><td>Safety: Random faults in a single channel are masked
Availability: The full system functionality is still available in case of a single random fault
Maintenance: Hardware channels can be maintained at runtime</td><td>Triple hardware costs for system replication</td></tr>
<tr><td>Credits</td><td colspan="2">[Douglass 2002] formulates this well-known architecture as a pattern. [Armoush 2010] adds detailed information about quality attribute related consequences.</td></tr>
</table>

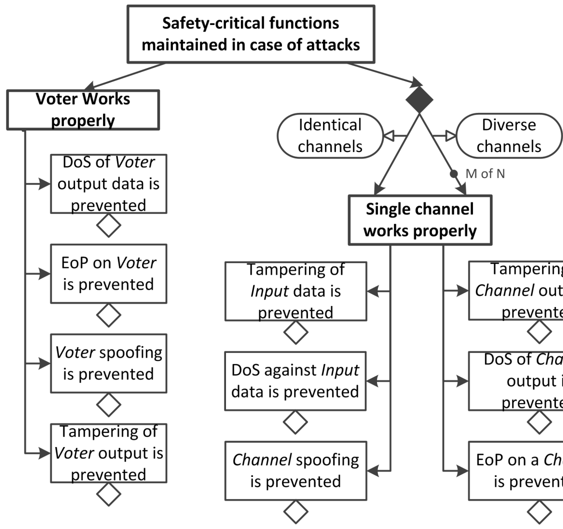
Pattern Name	M-out-of-N Pattern — **Pattern Type**: hardware/software, fail-over
Also Known As	M/N Parallel Redundancy Pattern, MooN Pattern
Context	A safety-critical application without a fail-safe state has a high random error rate and a low or high systematic error rate.
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components.
Forces	- the system cannot shut down because it has no safe state - high safety certification levels require handling of systematic faults - the safety standard requires high fault coverage for single-point of failure components - high availability requires hardware platforms to be maintained at the runtime
Solution	N identical or diverse channels (software or hardware) operate in parallel. If a fault occurs in one channel then the other channels still produce the correct output. A voter decides for the result given by at least M channels.



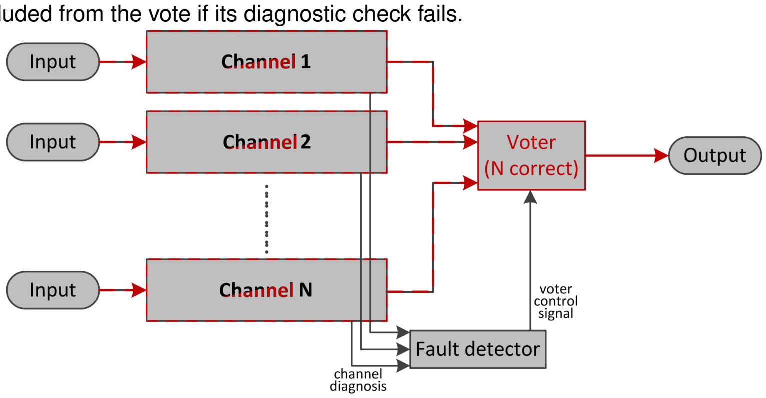
Security GSN

The voter is a single point of failure of this pattern and has to be well protected against security flaws.
* For MooN systems with diverse channels, attacks on a single channel are not critical. For MooN systems using identical channels, attacks on a single channel are critical, because if an attacker can compromise a single channel, can also compromise other identical channels without a lot of effort.

	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of *Voter* Spoofing of *Channel**	-	Spoofing of *Channel**
T	Tampering of *Voter* output Tampering of *Channel* input* Tampering of *Channel* output*	-	Tampering of *Channel* input* Tampering of *Channel* output*
R	-	-	-
I	-	-	-
D	DoS of *Voter* output data DoS of *Channel* input* DoS of *Channel* output*	-	DoS of *Channel* input* DoS of *Channel* output*
E	EoP on *Voter* EoP on *Channel**	-	EoP on *Channel**

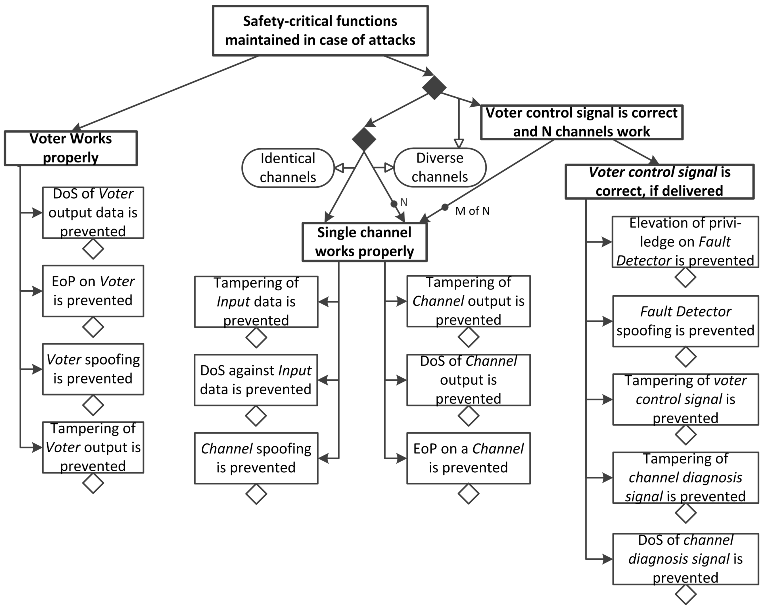
<table>
<tr><td></td><td colspan="2">
</td></tr>
<tr><td rowspan="4">Consequences</td><td colspan="2">This pattern does not identify the type or the reason of the fault; it just determines the module that contains a fault without correcting the fault itself. To achieve high reliability, the voter has to be very reliable.</td></tr>
<tr><td colspan="2">Affected Attributes</td></tr>
<tr><td>Positively</td><td>Negatively</td></tr>
<tr><td>Safety: Single-channel random or systematic faults are masked
Availability: The full system functionality is still available in case of a single fault
Maintenance: Hardware channels can be maintained at runtime</td><td>Multiple hardware costs for system replication and multiple development costs if diverse channels are used</td></tr>
<tr><td>Credits</td><td colspan="2">[Grunske 2003] describes this pattern and calls it MULTI-CHANNEL-REDUNDANCY WITH VOTING. [Armoush 2010] adds detailed information about quality attribute related consequences.</td></tr>
</table>

Pattern Name	M-OUT-OF-N-D PATTERN
Pattern Type	hardware/software, fail-over
Also Known As	MooN-D Pattern
Context	A safety-critical application without a fail-safe state has a high random error rate and a low or high systematic error rate.
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components.
Forces	- the system cannot shut down because it has no safe state - high safety certification levels require handling of systematic faults - the safety standard requires high fault coverage for single-point of failure components - due to these high availability requirements the hardware platforms must be maintained at the runtime of the system
Solution	N identical or diverse channels operate in parallel. If a single fault occurs in one channel then the other channels still produce the correct output. A *Voter* decides for the result given by at least M channels. The *Voter* can be influenced by a diagnostic check implemented within the channels. For example, a channel could be excluded from the vote if its diagnostic check fails.



Security GSN	The voter is a single point of failure of this pattern and has to be well protected against security flaws. * For MooN systems with diverse channels, attacks on a single channel are not critical. For MooN systems using identical channels, attacks on a single channel are critical, because if an attacker can compromise a single channel, can also compromise other identical channels without a lot of effort.

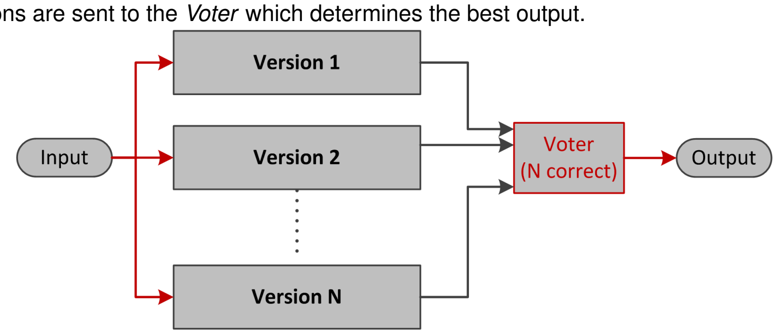
	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of *Voter* Spoofing of *Channel**	-	Spoofing of *Channel**
T	Tampering of *Voter* output Tampering of *Channel* input* Tampering of *Channel* output*	-	Tampering of *Channel* input* Tampering of *Channel* output*
R	-	-	-
I	-	-	-
D	DoS of *Voter* output data DoS of *Channel* input* DoS of *Channel* output*	-	DoS of *Channel* input* DoS of *Channel* output*
E	EoP on *Voter* EoP on *Channel**	-	EoP on *Channel**

		
Consequences	This pattern can identify the type or the reason of a fault. System reliability strongly depends on the voter unit, the diagnostic test, and the level of diversity between the two channels.	
	Affected Attributes	
	Positively	**Negatively**
	Safety: Random or systematic faults in a single channel are masked *Availability:* The full system functionality is still available in case of a single fault *Maintenance:* Hardware channels can be maintained at runtime	Multiple hardware costs for system replication and multiple development costs if diverse channels are used
Credits	The MooN-D architecture is described by the IEC 61508 standard.	

Pattern Name	N-Version Programming Pattern	**Pattern Type**	software, fail-over
Also Known As	-		
Context	A safety-critical software without a fail-safe state which probably contains software faults.		
Problem	How to design a system which continues operating even in the presence of software faults.		
Forces	- software often contains faults - high safety certification levels require handling of systematic faults - the safety standard requires high fault coverage for single-point of failure components		

Solution

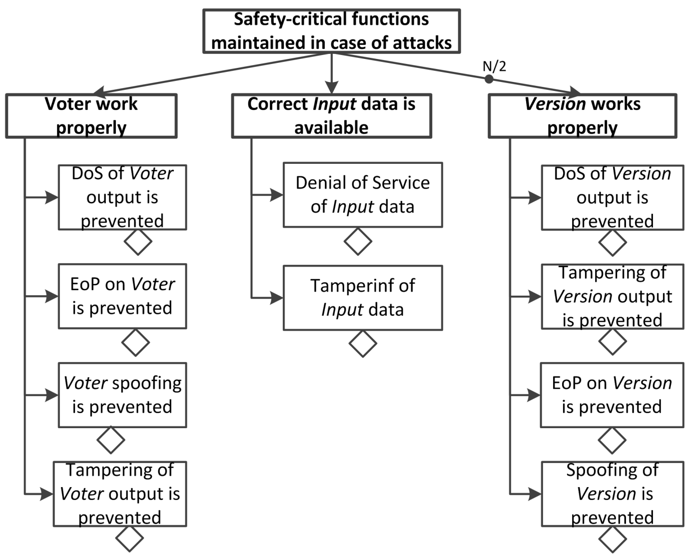
N software versions are developed independently from the same initial specification. The outputs of these versions are sent to the *Voter* which determines the best output.



Security GSN

The voter and the single input source are single points of failure of this pattern and have to be well protected against security flaws.

	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of the *Voter*		Spoofing of a *Version*
T	Tampering of *Version* input Tampering of *Voter* output		Tampering of *Version* output
R	-		
I	-		
D	DoS of *Version* input DoS of *Voter* output		DoS of *Version* output
E	EoP of *Voter*		EoP of *Version*

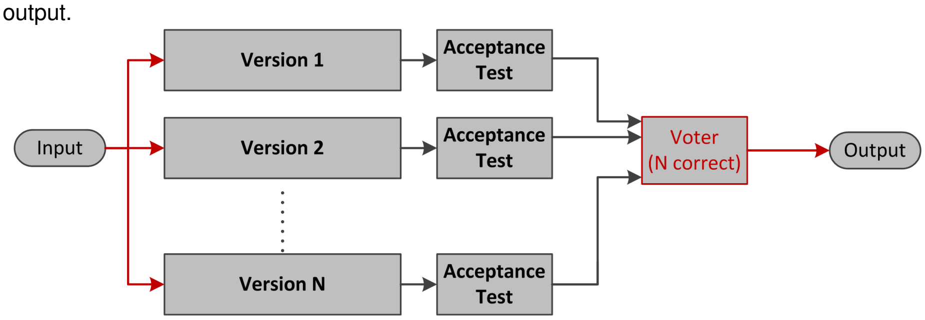


<table>
<tr><td rowspan="4">Consequences</td><td colspan="2">This pattern can handle systematic faults in the software. A drawback is that the high dependency on the initial specification may lead to a propagation of dependent faults to all versions. The voter has to be highly reliable.</td></tr>
<tr><td colspan="2">Affected Attributes</td></tr>
<tr><td>Positively</td><td>Negatively</td></tr>
<tr><td>Safety: Software faults are handled but not detected
Availability: The full system functionality is still available in case of faults</td><td>Costs: Multiple development costs, multiple hardware costs if the software versions run on separate hardware
Modifications: Multiple effort for software modifications</td></tr>
<tr><td>Credits</td><td colspan="2">[Armoush 2010] presents this pattern with detailed information about quality attribute related consequences.</td></tr>
</table>

Pattern Name	ACCEPTANCE VOTING PATTERN	**Pattern Type**	software, fail-over
Also Known As	-		
Context	A safety-critical software without a fail-safe state which probably contains software faults.		
Problem	How to design a system which continues operating even in the presence of software faults.		
Forces	- software often contains faults - high safety certification levels require handling of systematic faults - the safety standard requires high fault coverage for single-point of failure components		

Solution

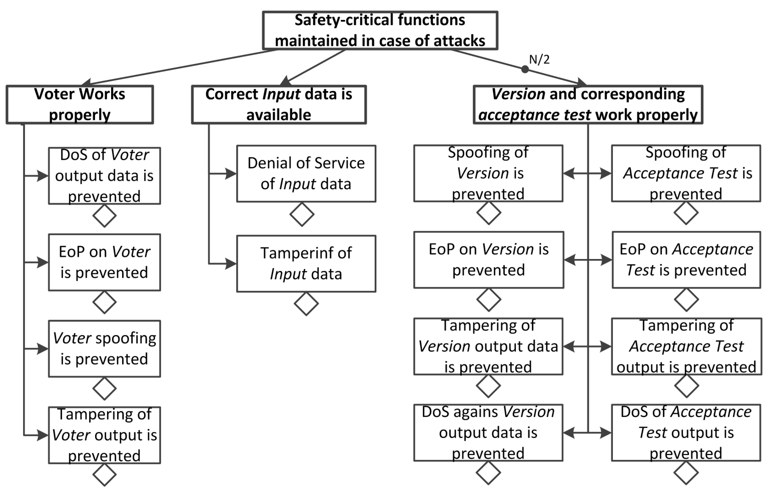
N software versions are developed independently from the same initial specification. The outputs of these versions are checked by an *Acceptance Test* and valid outputs are sent to a *Voter* which determines the best output.



Security GSN

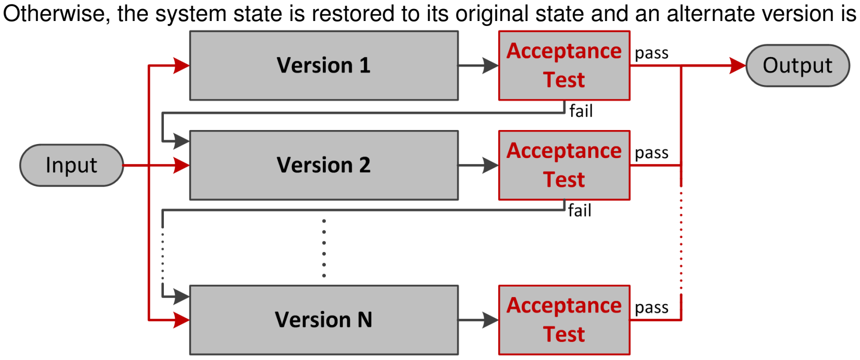
The voter and the single input source are single points of failure of this pattern and have to be well protected against security flaws.

	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of the *Voter*	-	Spoofing of a *Version* Spoofing of an *Acceptance Test*
T	Tampering of *Version* input Tampering of *Voter* output	-	Tampering of *Version* output Tampering of *Acceptance Test* output
R	-	-	-
I	-	-	-
D	DoS of *Version* input DoS of *Voter* output	-	DoS of *Version* output DoS of *Acceptance Test* output
E	EoP of *Voter*	-	EoP of *Version* EoP of *Acceptance Test*

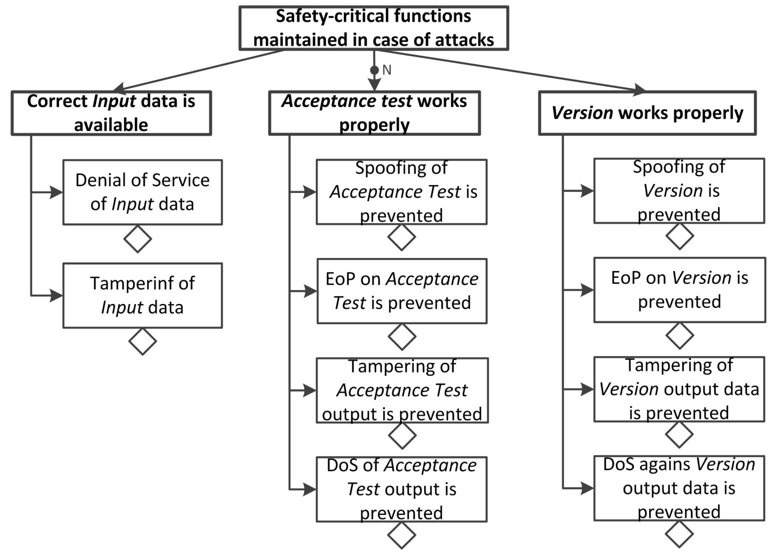


Consequences	This pattern can handle systematic faults in the software. A drawback is that the high dependency on the initial specification may lead to a propagation of dependent faults to all versions.	
	Affected Attributes	
	Positively	**Negatively**
	Safety: Software faults are handled and probably detected *Availability:* The full system functionality is still available in case of faults	*Costs:* Multiple development costs, multiple hardware costs if the software versions run on separate hardware *Modifications:* Multiple effort for software modifications
Credits	[Armoush 2010] presents this pattern with detailed information about quality attribute related consequences.	

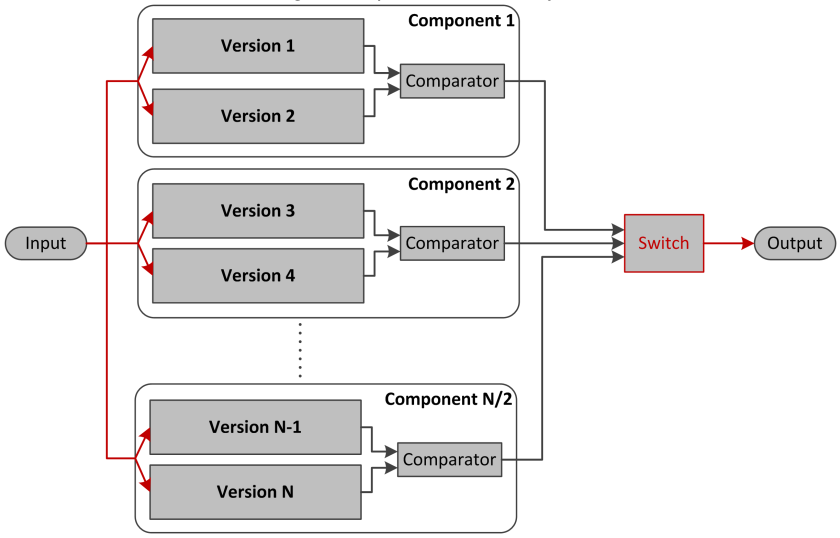
Pattern Name	RECOVERY BLOCK PATTERN	**Pattern Type**	software, fail-over
Also Known As	-		
Context	A safety-critical software without a fail-safe state which probably contains software faults.		
Problem	How to design a system which continues operating even in the presence of software faults.		
Forces	- software often contains faults - high safety certification levels require handling of systematic faults - the safety standard requires high fault coverage for single-point of failure components - no additional processing hardware or processing time is available		
Solution	N software versions are developed independently from the same initial specification. Only a single version is executed at a time. After the execution of *Version 1*, an *Acceptance Test* is executed to check if the software output is reasonable. If the *Acceptance Test* is passed, then the outcome is considered as correct. Otherwise, the system state is restored to its original state and an alternate version is invoked.		
Security GSN	The versions share a single input which has to be protected against attacks. Each acceptance test directly influences the final output. Therefore an attack on a single acceptance test can compromise the system.		



	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of an *Acceptance Test*	-	Spoofing of a *Version*
T	Tampering of *Version* input Tampering of *Acceptance Test* output	-	Tampering of *Version* output
R	-	-	-
I	-	-	-
D	DoS of *Version* input DoS of *Acceptance Test* output	-	DoS of *Version* output
E	EoP of *Acceptance Test*	-	EoP of *Version*

		
Consequences	This pattern can handle systematic faults in the software. A drawback is that the high dependency on the initial specification may lead to a propagation of dependent faults to all versions. Also the reliability highly depends on the quality of the acceptance test.	
	Affected Attributes	
	Positively	**Negatively**
	Safety: Software faults are handled and probably detected *Availability:* The full system functionality is still available in case of faults	*Costs:* Multiple development costs *Modifications:* Multiple effort for software modifications
Credits	[Armoush 2010] presents this pattern with detailed information about quality attribute related consequences.	

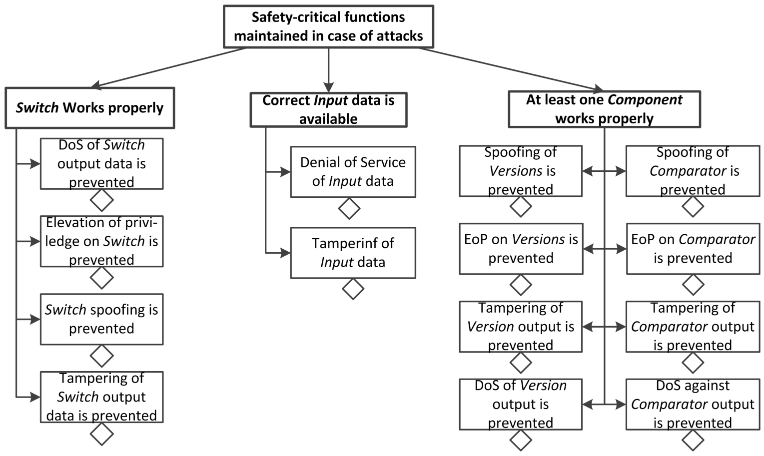
Pattern Name	N-SELF CHECKING PROGRAMMING PATTERN	Pattern Type	software, fail-over
Also Known As	-		
Context	A safety-critical software without a fail-safe state which probably contains software faults.		
Problem	How to design a system which continues operating even in the presence of software faults.		
Forces	- software often contains faults - high safety certification levels require handling of systematic faults - the safety standard requires high fault coverage for single-point of failure components		
Solution	N>=4 software versions are developed independently from the same initial specification. The versions are arranged in pairs of two as components. Within a component, the results of the two versions are compared to detect errors. If a component fails due to different results from its versions, the next component is invoked to start delivering the required functionality.		



Security GSN

To compromise the system an attacker can attack the switch or the input data which are both single points of failure. An attacker can also attack the components. The system is compromised if either the two versions in the first component produce the same wrong output or all versions produce any wrong output.

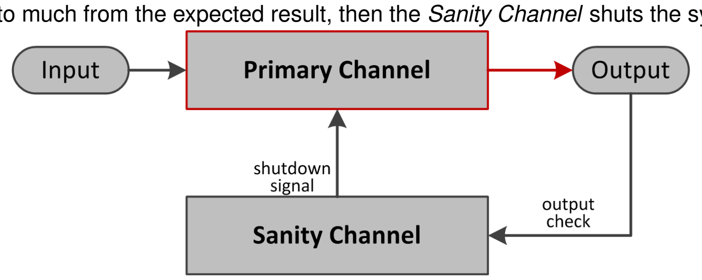
	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of *Switch*	-	Spoofing of a *Version* Spoofing of a *Comparator*
T	Tampering of *Version* input Tampering of *Switch* output	-	Tampering of *Version* output Tampering of *Comparator* output
R	-	-	-
I	-	-	-
D	DoS of *Version* input DoS of *Switch* output	-	DoS of *Version* output DoS of *Comparator* output
E	EoP of *Switch*	-	EoP of *Version* EoP of *Comparator*

<table>
<tr><td></td><td colspan="2">
</td></tr>
<tr><td rowspan="4">Consequences</td><td colspan="2">This pattern can handle systematic faults in the software. A drawback is that the high dependency on the initial specification may lead to a propagation of dependent faults to all versions.</td></tr>
<tr><td colspan="2">Affected Attributes</td></tr>
<tr><td>Positively</td><td>Negatively</td></tr>
<tr><td>Safety: Software faults are handled and probably detected at component level
Availability: The full system functionality is still available in case of faults</td><td>Costs: Multiple development costs
Modifications: Multiple effort for software modifications</td></tr>
<tr><td>Credits</td><td colspan="2">[Armoush 2010] presents this pattern with detailed information about quality attribute related consequences.</td></tr>
</table>

Pattern Name	SANITY CHECK PATTERN	**Pattern Type**	hardware, fail-safe
Also Known As	-		
Context	A safety-critical system with a fail-safe state and low availability requirements.		
Problem	Find an appropriate mechanism to detect failures or errors that can lead to known hazards.		
Forces	- The set of relevant hazards is often known for a specific application domain - Full redundancy solutions are expensive		

Solution

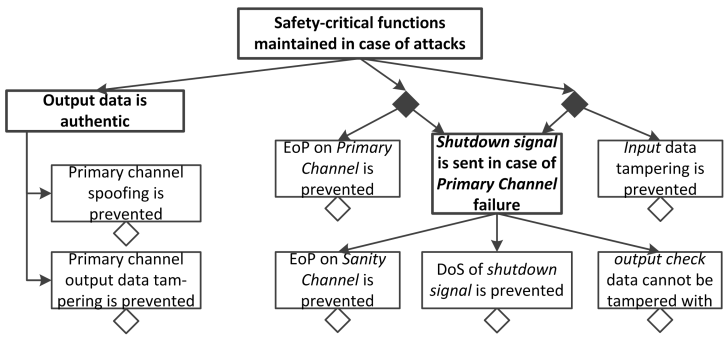
A separate *Sanity Channel* monitors the correct operation of the *Primary Channel*. If the *Primary Channel* output deviates to much from the expected result, then the *Sanity Channel* shuts the system down.



Security GSN

Most of the attacks on the system lead to a safe state. The only single attack which can compromise the system safety is if an attacker can directly modify the output of the system.

	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of *Primary Channel*	Spoofing of *Sanity Channel*	-
T	Tampering of *Primary Channel* output	Tampering of *Primary Channel* input Tampering of *shutdown signal* Tampering of *output check*	-
R	-	-	-
I	-	-	-
D	-	DoS of *Primary Channel* input DoS of *Primary Channel* output DoS of *output check*	DoS of *shutdown signal*
E	-	EoP on *Primary Channel* EoP on *Sanity Channel*	-



Consequences

The sanity channel is diverse from the primary channel which allows limited systematic fault as well as random fault detection.

Affected Attributes	
Positively	**Negatively**
Safety: Known hazards can be handled	*Availability:* Decreased if the system goes into its safe state

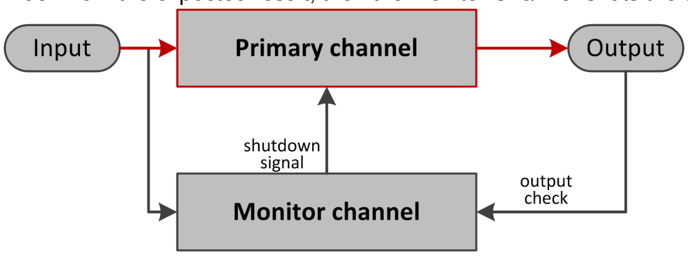
Credits

[Douglass 2002] introduces the pattern. [Grunske 2003] presents a more general version of this pattern and [Armoush 2010] adds detailed information about quality attribute related consequences.

Pattern Name	MONITOR-ACTUATOR PATTERN	**Pattern Type**	hardware, fail-safe
Also Known As	-		
Context	A safety-critical system with a fail-safe state and with low availability requirements.		
Problem	Find an appropriate mechanism to detect failures or errors		
Forces	- Full redundancy solutions are expensive		

Solution

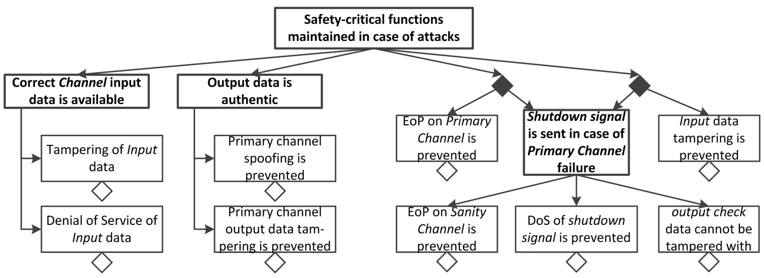
A separate *Monitor Channel* monitors the correct operation of the primary channel. The *Monitor Channel* computes reference values from the inputs and compares them to the *Primary Channel* output. If the value deviates to much from the expected result, then the *Monitor Channel* shuts the system down.



Security GSN

Most of the attacks on the system lead to a safe state. The only single attack which can compromise the system safety is if an attacker can directly modify the output or the input of the system.

	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of *Primary Channel*	Spoofing of *Monitoring Channel*	-
T	Tampering of *Primary Channel* output Tampering of *Channel* input	Tampering of *shutdown signal* Tampering of *output check* data	-
R	-	-	-
I	-	-	-
D	DoS of *Channel* input	DoS of *Primary Channel* output DoS of *output check* data	DoS of *shutdown signal*
E	-	EoP on *Primary Channel* EoP on *Monitor Channel*	-



Consequences

The monitor channel is diverse from the primary channel which allows limited systematic fault as well as random fault detection.

Affected Attributes	
Positively	**Negatively**
Safety: Hazardous situations can be detected and handled	*Availability:* Decreased if the system goes into its safe state

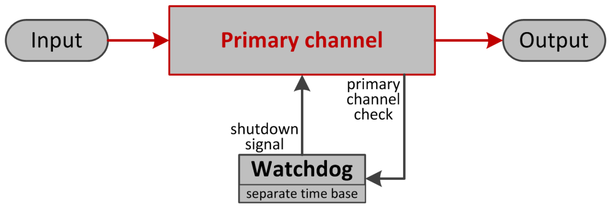
Credits

[Douglass 2002] introduces the pattern. [Grunske 2003] presents a more general version of this pattern and [Armoush 2010] adds detailed information about quality attribute related consequences.

Pattern Name	WATCHDOG PATTERN	Pattern Type	hardware, fail-safe
Also Known As	Watchdog Timer, Watchdog Processor, Hardware Watchdog Pattern		
Context	A system provides a timing-critical safety functionality.		
Problem	How to make sure that the internal computational processing is proceeding properly and timely.		
Forces	- Full redundancy solutions are expensive - An unavailable component cannot tell that it is unavailable		

Solution

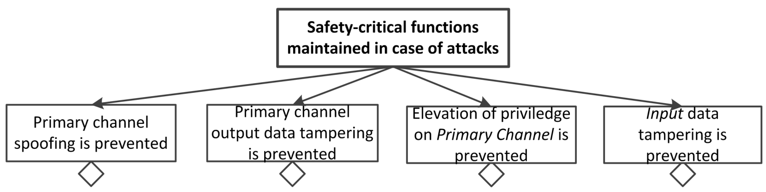
A separate *Watchdog* hardware component receives liveness messages from the *Primary Channel*. If the *Watchdog* does not receive the expected messages, it will initiate a corrective action such as a shutdown signal.



Security GSN

The system can be brought into a critical state by the same attacks as the simple system just consisting of a primary channel. The watchdog system is in safe-state if shut off. Therefore, DoS attacks are not safety-critical.

	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of *Primary Channel*	Spoofing of the *Watchdog*	-
T	Tampering of *Primary Channel* output Tampering of *Primary Channel* input	Tampering of *shutdown signal* Tampering of *primary channel check*	-
R	-	-	-
I	-	-	-
D	-	DoS of *Primary Channel* input DoS of *Primary Channel* output DoS of *primary channel check*	DoS of *shutdown signal*
E	EoP on *Primary Channel*	EoP on *Watchdog*	-



Consequences

The watchdog can detect failures of the primary channel if the failure affects the liveness messages. The watchdog is diverse from the primary channel which allows limited systematic fault detection.

Affected Attributes

Positively	Negatively
Safety: Timing faults are handled	*Availability:* Decreased if the system shuts down

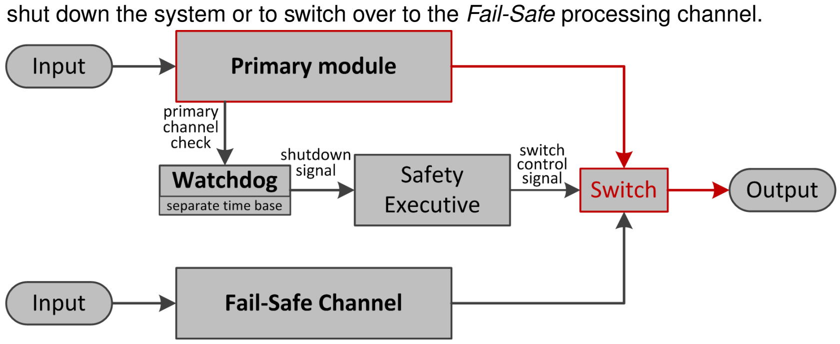
Credits

[Douglass 2002] introduces the pattern. [Grunske 2003] and [Hanmer 2007] also present this pattern and [Armoush 2010] adds detailed information about quality attribute related consequences.

Pattern Name	SAFETY EXECUTIVE PATTERN	**Pattern Type**	hardware, fail-safe
Also Known As	Safety Kernel Pattern, Shadow-Pattern, Simplex-Pattern		
Context	A system with a complex fail-safe state should maintain its safety functionality even in case of faults.		
Problem	How to check if a fail-safe state should be entered and how to maintain it.		
Forces	- Full redundancy solutions are expensive - An unavailable component cannot tell that it is unavailable - Complex fail-safe state		

Solution

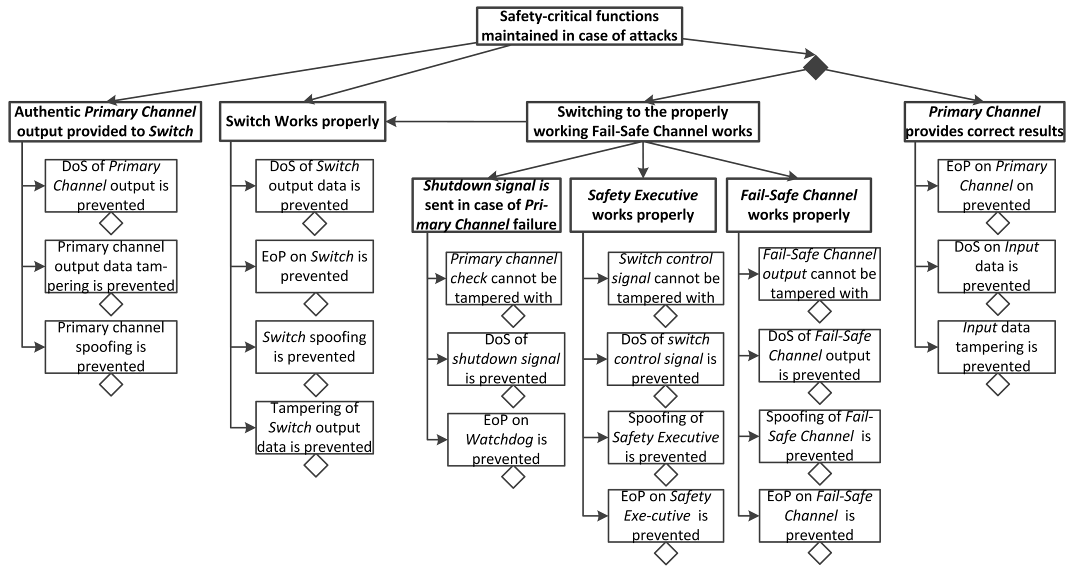
The *Primary Channel* performs all the required functionality. An optional *Fail-Safe Channel* executes just the safety-critical functionality. A centralized *Safety Executive* component coordinates all safety-measures required to shut down the system or to switch over to the *Fail-Safe* processing channel.



Security GSN

The switch is a single point of failure of this pattern and has to be well protected against security flaws.

	safety-critical	leads to a safe state	system remains fully functional
S	*Primary Channel* spoofing *Switch* spoofing	Spoofing of the *Watchdog* Spoofing of *Safety Executive*	Spoofing of *Fail Safe Channel*
T	Tampering of *Primary Channel* output Tampering of *Switch* output	Tampering of *Primary Channel* input Tampering with *shutdown signal* Tampering of *primary channel check* Tampering of *switch control signal*	Tampering of *Fail Safe Channel* output
R	-	-	-
I	-	-	-
D	DoS of *Switch* output DoS of *Primary Channel* output	DoS of *Primary Channel* Input DoS of *primary channel check*	DoS of *shutdown signal* DoS of *Fail Safe Channel* output DoS of *switch control signal*
E	EoP on *Switch*	EoP on *Primary Channel* EoP on *Watchdog* EoP on *Safety Executive*	EoP on *Fail Safe Channel*

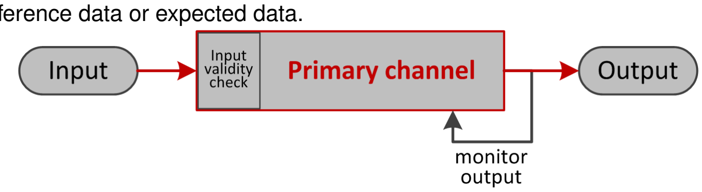


Consequences	The fail-safe processing channel is diverse from the primary channel which allows limited systematic fault detection.	
	Affected Attributes	
	Positively	**Negatively**
	Safety: A safe-state is entered if a fault occurs	*Availability:* Decreased if the system goes into its safe state
Credits	[Douglass 2002] introduces the pattern and [Armoush 2010] adds detailed information about quality attribute related consequences.	

Pattern Name	PROTECTED SINGLE CHANNEL	**Pattern Type**	hardware/software, fail-safe
Also Known As	Safety Kernel Pattern, Shadow-Pattern, Simplex-Pattern		
Context	A system with a fail-safe state and with low availability requirements.		
Problem	Find an appropriate mechanism to handle failures or errors that can lead to known hazards.		
Forces	- Full redundancy solutions are expensive - Components have become so complex that we cannot assume them to be error free - Not any additional hardware components can be introduced		

Solution

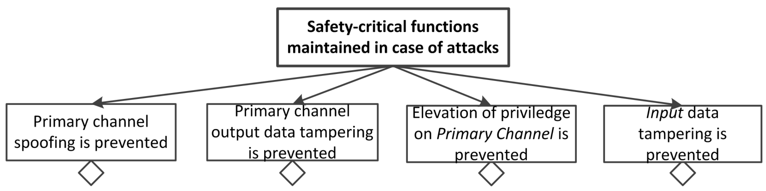
The input and/or output data of the *Primary Channel* is monitored and checked regarding its validity or compared to reference data or expected data.



Security GSN

The system can be brought into a critical state by the same attacks as the simple system just consisting of a primary channel.

	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing if *Primary Channel*	-	-
T	Tampering of *Primary Channel* output Tampering of *Primary Channel* input	Tampering of *monitor output* signal	-
R	-	-	-
I	-	-	-
D	-	DoS of *Primary Channel* input DoS of *Primary Channel* output DoS of *monitor output* signal	-
E	EoP on *Primary Channel*	-	-



Consequences

The checks are diverse from the primary channel functionality which allows limited systematic fault detection.

Affected Attributes	
Positively	**Negatively**
Safety: Known hazards can be handled	*Availability:* Decreased if the system goes into its safe state

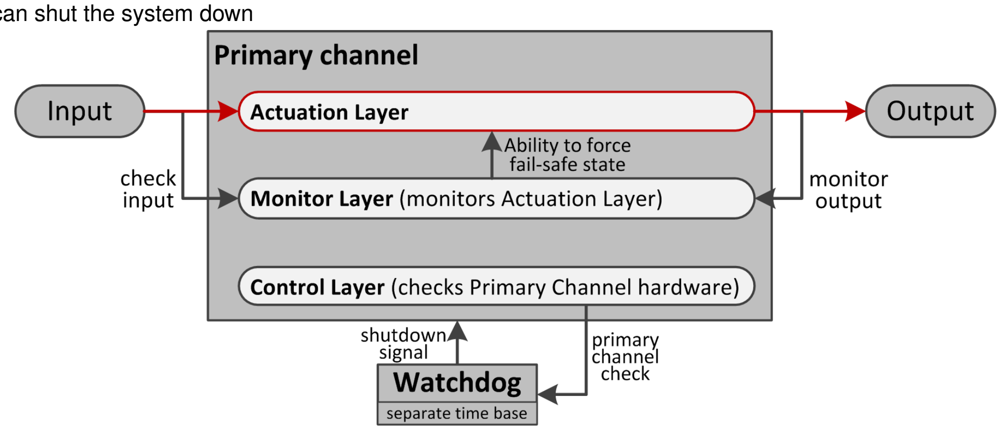
Credits

[Douglass 2002] introduces the pattern. [Grunske 2003] also presents this pattern and [Armoush 2010] adds detailed information about quality attribute related consequences.

Pattern Name	3-Level Safety Monitoring	**Pattern Type**	hardware/software, fail-safe
Also Known As	Safety Kernel Pattern, Shadow-Pattern, Simplex-Pattern		
Context	A system with a fail-safe state and with low availability requirements		
Problem	Find an appropriate mechanism to handle failures or errors that can lead to known hazards.		
Forces	- Full redundancy solutions are expensive - Components have become so complex that we cannot assume them to be error free		

Solution

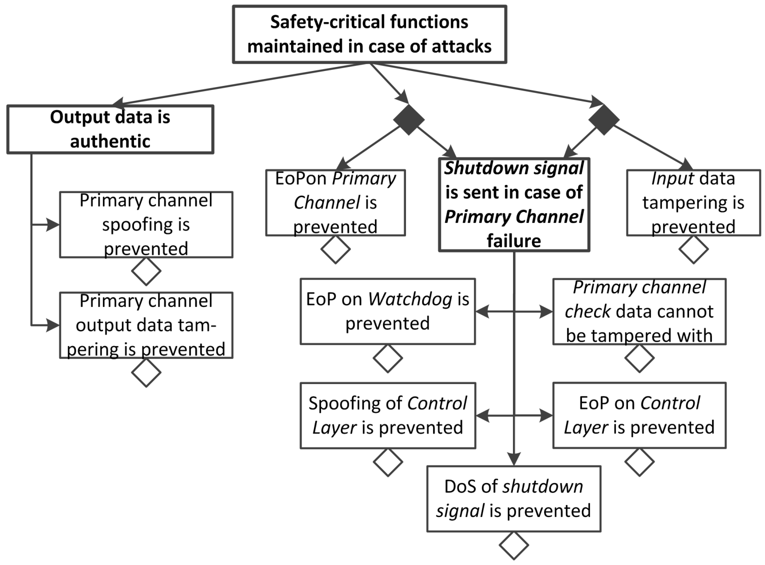
Divide the system into 3 layers:
- The *Actuation Layer* performs the system functionality
- The *Monitoring Layer* monitors the *Actuation Layer* and forces a fail-safe state if values deviate too much from references
- The *Control Layer* checks the system hardware and sends messages to a *Watchdog* component which can shut the system down



Security GSN

Most of the attacks on the system lead to a safe state. The only single attack which can compromise the system safety is if an attacker can directly modify the input or output of the system.

	safety-critical	leads to a safe state	system remains fully functional
S	Spoofing of the *Actuation Layer*	Spoofing of the *Watchdog* Spoofing of the *Monitor Layer* Spoofing of the *Control Layer*	-
T	Tampering of *Input* data Tampering of *Actuation Layer* output	Tamper with *shutdown signal* Tampering of *primary channel check* Tampering of *Monitor Layer* output	-
R	-	-	-
I	-	-	-
D	DoS of *Input* data DoS of *Actuation Layer* output	DoS of *primary channel check* DoS of *Monitor Layer* output	DoS of *shutdown signal*
E	-	EoP on *Actuation Layer* EoP on *Watchdog* EoP on *Monitor Layer* EoP of *Control Layer*	-

		
Consequences	The pattern is not applicable for systems with high availability requirements. The checks are diverse from the primary channel which allows limited systematic fault detection.	
	Affected Attributes	
	Positively	**Negatively**
	Safety: Known hazards can be handled	*Availability:* Decreased if the system goes into its safe state
Credits	[Armoush 2010] presents this pattern with detailed information about quality attribute related consequences.	