Cybersecurity Guidelines for Software Development & Assessment

BV-SW-200 / 20170609

Move Forward with Confidence

BUREAU VERITAS


TARGET
TARGET
TARGET
TARGET
TARGET

The Cybersecurity Guidelines for Software Development and Assessment, as well as all information included within, are protected by copyright and are the exclusive property of Bureau Veritas. These Cybersecurity Guidelines are meant to be a freely downloadable document. However, and notwithstanding anything to the contrary, all intellectual property rights related to this document including but not limited to the names, service marks, trademarks, inventions, logos and copyrights of Bureau Veritas and its affiliates, are and shall remain the sole property of Bureau Veritas or its affiliates and shall not be used by any person or entity, except solely to the extent that this person or entity obtains the prior written approval of Bureau Veritas and then only in the manner prescribed by Bureau Veritas.

No part of this document shall be modified in any form and by any means in any part of the world, without the prior written consent of Bureau Veritas. In particular, Bureau Veritas cannot be held liable for any update, modification or other amendment or alteration of this document by any person or entity for any reason whatsoever. No person or entity using this document shall contest the validity of the rights or take any action that might impair the value or goodwill associated with the marks or the image or reputation of Bureau Veritas or its affiliates. Any person or entity downloading or using this document shall take all necessary steps to ensure that it operates at all times in accordance with all applicable data protection laws and regulations.

In no event shall Bureau Veritas, its agents, consultants, and subcontractors, be liable for special, indirect or consequential damages resulting from or arising out of the use of these Cybersecurity Guidelines, including, without limitation, loss of profit or business interruptions, however these may be caused.

The user shall indemnify and hold harmless Bureau Veritas against any and all claims from third parties arising from or in connection to its use of this document.

Every effort is made to provide general information. However, Bureau Veritas does not guarantee the accuracy, completeness, adequacy or usefulness of the content of the document, including but not limited to, any information, product, service or process disclosed herein. Bureau Veritas hereby disclaims all warranties and guarantees, whether expressed or implied, including any warranty of merchantability, fitness for a particular purpose or use, or non-infringement of third party rights with respect to the documents provided.

Copyright © 2017 BUREAU VERITAS, All rights reserved.
Published by BUREAU VERITAS SA.
Co-written by BUREAU VERITAS SA and List, an institute of CEA Tech.

list
cea tech

BUREAU VERITAS
1828
BUREAU
VERITAS

CONTENTS

INTRODUCTION

1. Introduction

Software components are being used, at scale, to provide intelligent functionalities in complex systems. This ubiquity, combined with a sharp rise in cyber-threats, amplifies the need to identify clear and effective security practices for the development and assessment of software components.

NOTE: in the following, the definition of words written in UPPER CASE LETTERS is given in the Definitions section.

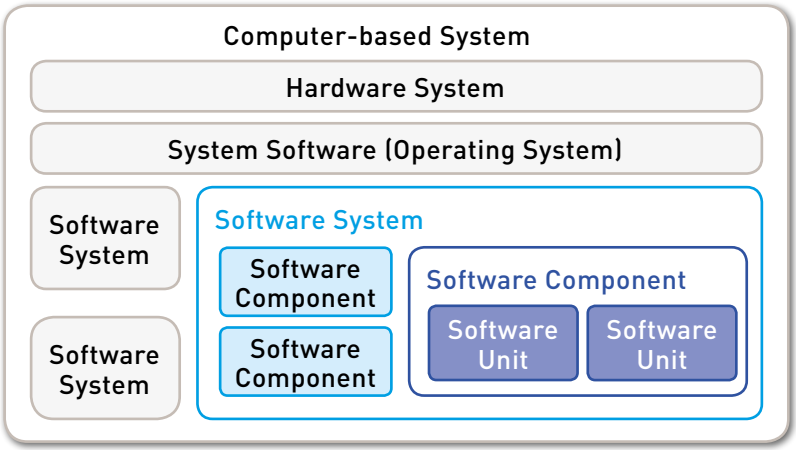


Figure 1. Generic computer-based system breakdown

1.1. Purpose of these guidelines

This document describes a list of objectives to develop, verify, and operate a SOFTWARE SYSTEM that satisfies an intended level of SECURITY. This includes:

- the CONFIDENTIALITY and INTEGRITY of the data processed by the system,
- the AVAILABILITY of the service provided by the system.

The target audience of this document is mainly (but is not restricted to): software developers, product managers, security teams, quality assessors and software security auditors/assessors.

The distinctive feature of these guidelines is the focus on the use of SOFTWARE SUPPORT TOOLS to satisfy the SOFTWARE SYSTEM development and assessment objectives, especially for the analysis of the program structure and CODE.

The level of SECURITY required by the SOFTWARE SYSTEM, and thus the scope and strength of its objectives, is determined through a risk-based methodology.

The present guidelines complement the BV-SW-100 guidelines on software performance («Software Development & Assessment Guidelines»), and can be deployed accordingly. The use of the BV-SW-100 is in particular helpful as a reference for availability (reliability) guarantees.

This document was designed to help the target audience to develop secure mission-critical software in industrial control systems (e.g. embedded softwares in vehicles, connected objects, process control...). It can also be applied to consumer electronics, enterprise computing systems, and other software-intensive systems.

Within the scope of these guidelines, a certificate of compliance may be issued.

1.2. State of the art of software security assessment

A classical SECURITY assessment of a SOFTWARE SYSTEM is generally based on distinct activities:

- identification of the security threats in regards of the intended purpose of the SOFTWARE SYSTEM;
- independent evaluations of an organization's SECURITY, designed to test its defenses as a whole, including human factors;
- penetration tests and SECURITY audits: assessment of a digital system's SECURITY, by way of its resistance to attacks;
- vulnerability scans: monitoring of databases for publicly-known vulnerabilities that might impact a digital system;
- organizational audits: assessment of the SECURITY governance.

From the technical SECURITY foundations underlying these activities, the present guidelines build a concise list of objectives and acceptance criteria for SOFTWARE SYSTEMS, following the SECURITY by design concept. They leverage the demonstrated advantages of a trusted whitebox approach to set objectives on the development process and on some specific verification activities.

Current computing technologies (e.g. CODE analysis tools) can perform deep examinations of the program, efficiently supporting SOFTWARE SYSTEM verification activities. This document takes into account the use of those tools and their associated benefits, into a more general development and assessment strategy.

This approach is complementary to the performance assessment process from BV-SW-100. In particular safety processes and results demonstrably reduce the effort of SECURITY assessment activities.

1.3. Scope

SECURITY has to be managed holistically, and in particular SECURITY OBJECTIVES have to be divided between SOFTWARE SYSTEMS and hardware.

The present guidelines apply to SOFTWARE SYSTEMS involved in COMPUTER-BASED SYSTEMS, whatever their purpose of service. They cover the whole lifecycle of these systems, from specification to operation, maintenance and decommission.

The acceptable scope for the application of the guidelines is further discussed in Section 2.1 dealing with the system architecture analysis.

The results of the guidelines have to be completed with a hardware SECURITY analysis. Hardware analyses can be performed for instance based on IEC 15408 or IEC 62443. They usually encompass:

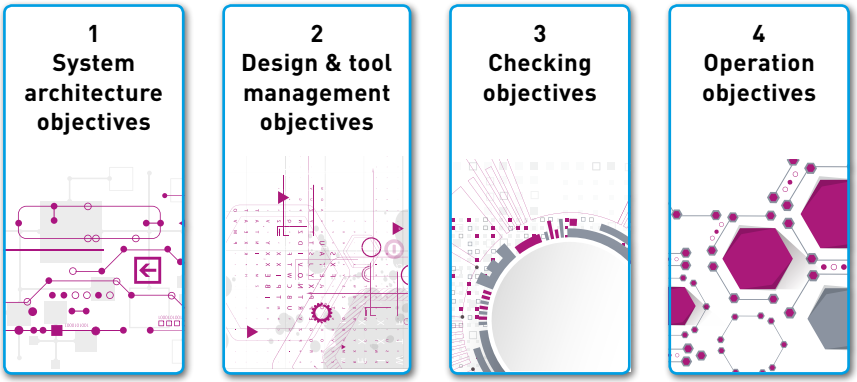
- the definition of the methodology;
- the identification of an evaluation perimeter;
- the realization of verification and testing activities;
- the summary of the results and the exported constraints on the system or software.

Therefore specific requirements of the interfaces between SOFTWARE SYSTEMS and hardware is out of the scope of the present guidelines.

The objectives to be assessed are summarized in the dedicated Checklist section (cf. appendix 1 page 30).

1.4. Guidelines structure

The structure of the guidelines is the following:



1.5. Definitions

- **AUTHENTICATION:** provision of assurance that a claimed characteristic of an entity is correct (ISO 27000).
- **AVAILABILITY:** property of being accessible and usable upon demand by an authorized entity (ISO 27000).
- **BLACK BOX TESTING:** testing, either functional or non-functional, without reference to the internal structure of the software system, software component or software unit.
- **CODE:** implementation of specific data on a specific computer program in a symbolic form, such as source code, object code, or executable code.
- **COMPROMISSION:** result of the occurrence of an un-mitigated SECURITY THREAT.
- **COMPUTER-BASED SYSTEM:** integrated set of hardware and software systems, including the system software, that provides the capability of satisfying a stated need or objective.
- **CONFIDENTIALITY:** property that information is not made available or disclosed to unauthorized individuals, entities, or processes (ISO 27000).
- **COTS** ("component off-the-shelf"): software component that already exists and is not developed specifically for the current project or computer-based system.
- **DETAILED DESIGN:** logic continuation of the preliminary design, considered as a process of modelling, usually with the express purpose of isolating one or more attribute(s) of the software, to prevent specific interactions and cross-coupling interference.
- **INFORMATION FLOW:** pattern in which information is passed within and between SOFTWARE UNITS, SOFTWARE COMPONENTS, and SOFTWARE SYSTEMS.
- **INTEGRITY:** property of safeguarding the accuracy and completeness of assets (ISO 27000).
- **FEARED EVENT:** undesired event that affects the reliability, availability, maintainability, safety and/or SECURITY of a computer-based system.
- **PENETRATION TEST:** specific TESTING activity where SECURITY OBJECTIVES are evaluated during the execution of a COMPUTER-BASED SYSTEM.
- **PROGRAMMABLE ELECTRONIC (COMPUTER-BASED):** device based on computer technology that includes hardware, software, input and output units.

  NOTE: this term covers microelectronics devices based on one or more central processing units (CPUs) together with associated memories, drivers, etc.
- **SDLC** (Security Development Life Cycle): process designed to increase resiliency and trustworthiness toward a product or a SOFTWARE SYSTEM.

- **SECURITY (CYBERSECURITY):** items related to external attacks driven by PROGRAMMABLE ELECTRONIC. SECURITY does not include SAFETY objectives.
- **SECURITY OBJECTIVES:** set of counter-measures to SOFTWARE THREATS.
- **SECURITY REQUIREMENTS:** set of properties of the TARGET OF EVALUATION that indicate how it meets its SECURITY OBJECTIVES.
- **SECURITY THREAT:** external FEARED EVENT that can affect the confidentiality, integrity, availability and/or authenticity of a SOFTWARE COMPONENT or SOFTWARE SYSTEM.
- **SOFTWARE CATEGORY:** classification of software systems and software components depending on the impact of feared events on the COMPUTER-BASED SYSTEM in which they are integrated.
- **SOFTWARE (SECURITY) CHECKING:** a set of activities organized to evaluate whether a SOFTWARE COMPONENT implementation satisfies a set of SECURITY REQUIREMENTS. SOFTWARE CHECKING encompasses STATIC ANALYSIS, TESTING, and PENETRATION TESTING.

  NOTE 1: in this document, SOFTWARE CHECKING is the process dedicated to the evaluation of a product, whereas VERIFICATION is the process dedicated to the evaluation of the activities that develop this product (i.e. process verification).
- **SOFTWARE COMPONENT:** any identifiable part of a computer program.

  NOTE: three terms identify the software decomposition. The top level is the software system. The lowest level that is not further decomposed is the software unit. Any level of composition, including the top and bottom levels, can be called a software component.
- **SOFTWARE SYSTEM:** integrated collection of software components organized to accomplish a specific function or set of functions.
- **SOFTWARE SUPPORT TOOL:** software tool that supports a phase of the software development lifecycle. Software tools may be divided into the following classes (IEC 61508):
  - T1 generates outputs that do not affect neither the executable code nor the data files of the software system;

    NOTE 1: T1 examples include text editors, requirements and design support tools without automatic code generation capabilities, and configuration tools.
  - T2 supports the test or verification of the design or executable code, where the tool may fail to detect defects, but cannot directly create errors neither in the executable code nor in the data files of the software system;

    NOTE 2: T2 examples include test harness generators, test coverage measurement tools, and static analysis tools.
  - T3 generates outputs that become part of the executable code and/or the data files of the software system.

    NOTE 3: T3 examples include code generators, compilers, linkers, and loaders.
- **SOFTWARE THREAT:** SECURITY THREAT that related specifically to a SOFTWARE SYSTEM, a SOFTWARE COMPONENT or a SOFTWARE UNIT.

- **SOFTWARE UNIT:** software component that is not subdivided into other components.
- **STATIC ANALYSIS:** subset of software checking activities where the software (i.e. a software unit, software component or software system) is not executed (e.g. code inspections, code analysis, software walkthroughs, software metrics, complexity analyses, etc.).
- **OPERATING SYSTEM** (or system software, also known as kernel): software component of the computer-based system that performs internal functions of the integrated programmable electronic device as opposed to the application software (software system) that provides external functions (as intended by an external user and/or device).
- **TESTING:** a subset of software checking activities where the software (i.e. a software unit, software component or software system) is executed to evaluate its actual behaviour (e.g. unit tests, integration tests, acceptance tests, etc.).
  NOTE: functional validation is achieved through testing.
- **TARGET OF EVALUATION:** SOFTWARE SYSTEM being assessed, together with its operational environment.
- **VERIFICATION:** process of examining the result of a given activity to determine its conformity with the stated objectives for that activity.
  NOTE: verification includes the common activities of requirement traceability.
- **WHITE BOX TESTING:** testing based on an analysis of the internal structure of the software system, software component or software unit.

1.6. Abbreviations

- **API:** Application Programming Interface
- **COTS:** Component Off-The-Shelf
- **CERT:** Computer Emergency Response Team
- **CRC:** Cyclic Redundancy Check
- **CVE:** Common Vulnerabilities and Exposures
- **CWE:** Common Weakness Enumeration
- **HMI:** Human-Machine Interface
- **REX:** Return on Experience
- **SC:** Software Category
- **SDLC:** Security Development Life Cycle
- **SR:** SECURITY REQUIREMENTS
- **TOE:** TARGET OF EVALUATION

1.7. How to read the objectives

The following parts of the document indicates the objectives to be met. Depending of the SOFTWARE CATEGORY (defined in section 2.1), objectives might be applicable or not.

For instance:

OBJECTIVES	ACCEPTANCE CRITERIA
OBJ_DES_080	**OBJ_DES_080**
Perform a cybersecurity oriented analysis on each SOFTWARE SUPPORT TOOL to ensure that it does not have an impact on the confidentiality, integrity & availability of the SOFTWARE SYSTEM. **[SC1]**	The analysis shall include a description of the interactions between the tools and the SOFTWARE SYSTEM (e.g. possible code modifications introduced by the tool). **[SC1]**

is an objective, identified OBJ_DES_080, required only if the SOFTWARE CATEGORY 1 [SC1] is targeted.

Left column : objectives
Right column : acceptance criteria

DEVELOPMENT & OPERATION OBJECTIVES AND ACCEPTANCE CRITERIA

2. Development & operation objectives and acceptance criteria

The objectives of this section are targeted at particular phases of the software lifecycle. The 4 main identified parts are:

- system architecture definition;
- design (including tools management);
- checking;
- operations.

The following scheme shows this organisation:

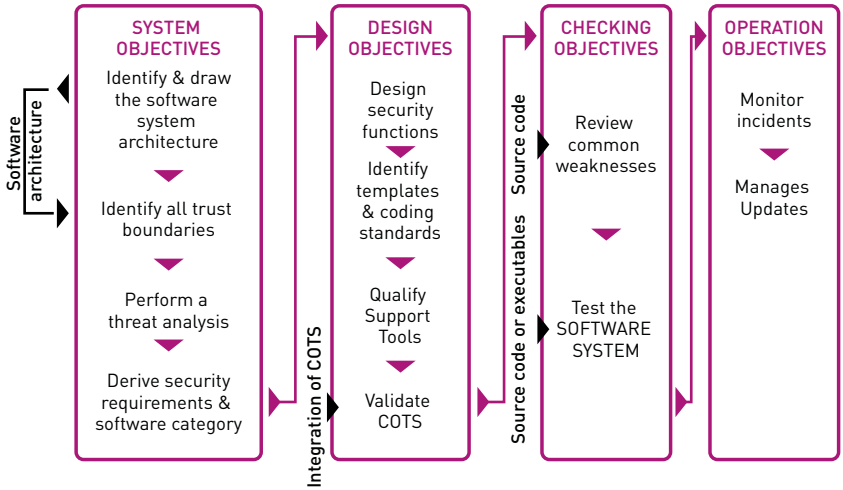
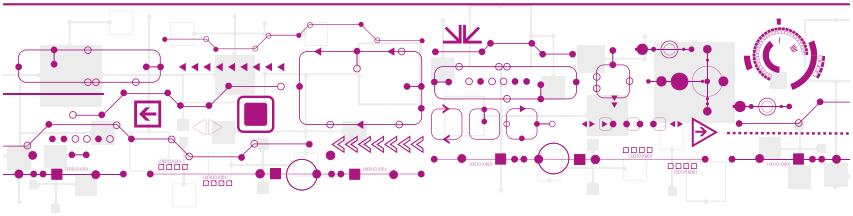


Figure 2. Global organisation of the software security management. (Credits: Bureau Veritas)

The definition of the software lifecycle has to be adapted to each SOFTWARE SYSTEM, based on the sequential software V cycle or within agile development methodologies.

The objectives documented hereunder are validated by one or more acceptance criteria. Objectives are meant as a high-level description of the desired functionalities. Acceptance criteria define the type of artefacts that can be used to provide clear and traceable guarantees of the correct implementation of desired functionalities.

2.1 System architecture



The COMPUTER-BASED SYSTEM may be subject, as part of FEARED EVENTS, to various SECURITY THREATS. It is therefore crucial to perform a threat analysis on each SOFTWARE COMPONENT at the beginning of the software development cycle, after the architecture design and the definition of the functional requirements of the SOFTWARE SYSTEM (this Design objective is covered under OBJ_SYS_020, page 17).

This analysis allocates a SOFTWARE CATEGORY to each of the SOFTWARE COMPONENTS or to the SOFTWARE SYSTEM, based on the number and severity of the identified threats.

The following SOFTWARE CATEGORIES table is given as an example, and can be extended depending on the maturity of an existing SDLC process and the granularity of security requirements:

SOFTWARE CATEGORY	Effects
0	Basic SECURITY requirements. Designed for a low overhead and easy integration of security requirements into an existing development process.
1	Advanced SECURITY requirements. It will require a bigger effort, but will result in heightened confidence in the SECURITY of critical software.

Table 1: Example of a SOFTWARE CATEGORY scale reference

OBJECTIVES

OBJ_SYS_010

Identify the SECURITY perimeter of the SOFTWARE SYSTEM and of each individual SOFTWARE COMPONENTS. This step allows the derivation of a threat model and the definition of a SOFTWARE CATEGORY.
[SC0,SC1]

ACCEPTANCE CRITERIA

OBJ_SYS_010

The perimeter includes:

- external and internal interfaces (e.g. between SOFTWARE COMPONENTS, between the SOFTWARE SYSTEM and hardware components)with their data type;
  [SC0,SC1]
- external entities allowed to perform actions on the SYSTEM and/or a SOFTWARE COMPONENT (e.g. users, other hardware or software systems);
  [SC0,SC1]
- external dependencies and COTS linked to the COMPONENT or the SYSTEM and their current SECURITY status;
  [SC0,SC1]
- data flow (e.g. functionalities call, network traffic, storage I/O) that an attacker may manipulate to COMPROMISE a SOFTWARE COMPONENT or the SYSTEM;
  [SC0,SC1]
- data stores (e.g. memory, shared memory, database) and control mechanisms that might be attacked by SECURITY THREATS;
  [SC0,SC1]
- trust boundaries between SOFTWARE COMPONENTS or SOFTWARE SYSTEMS.
  [SC0,SC1]

OBJECTIVES

OBJ_SYS_020

Perform system- and component-level threat analysis that includes defining a SOFTWARE CATEGORY for the SOFTWARE SYSTEM and performing threat analysis for the SOFTWARE COMPONENTS it includes. It is necessary to handle this analysis at the system level to take into account all the contributions of the hardware and software composing the system.

This analysis shall result in a categorization of the SOFTWARE COMPONENTS based on their SOFTWARE CATEGORY, and in a collection of identified threats and associated security requirements.

[SC0,SC1]

NOTE: the Appendix 2 - Example of threats identification & classification provides an example of such a threat analysis.

OBJ_SYS_030

Arbitrate performance and SECURITY trade-offs.

If performance and SECURITY requirements are incompatible for a given SOFTWARE COMPONENT, the design of the COMPUTER-BASED SYSTEMS shall:

- analyze the incompatibility;
- identify the risks due to either performance or SECURITY prioritization;
- when relevant, deport the discarded performance or SECURITY requirements onto another COMPONENT of the COMPUTER-BASED SYSTEM.

[SC0,SC1]

ACCEPTANCE CRITERIA

OBJ_SYS_020

The threat analysis can be done using an existing robust methodology (e.g.STRIDE, OCTAVE, Strike) or using a proprietary/in-house method.

It shall be based on the security perimeter previously identified and shall result in a documentation including identified threats and associated security requirements. When no security requirements are/can be proposed, the reasons shall be documented.

[SC0,SC1]

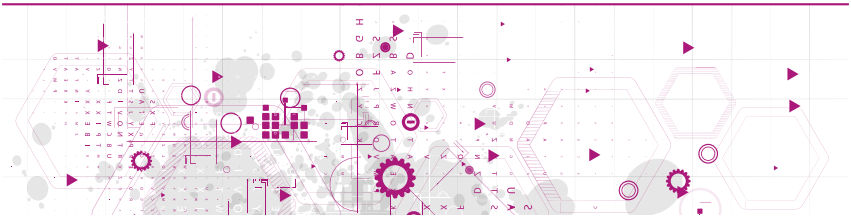
OBJ_SYS_030

The acceptance criteria shall examine:

- the documentation of the incompatibilities;
- the documentation of the arbitration and of its impact on the SOFTWARE SYSTEM.

[SC0,SC1]

2.2 Design & tool management



OBJECTIVES

OBJ_DES_010

Design and deploy adequate SECURITY functions.

In particular the SOFTWARE COMPONENT shall make use of existing SECURITY function inventories, such as the one from section 5.5 of ANSSI-CSPN-NOTE-01/2, part 2 of ISO/IEC 15408 or RGS Appendix B. This can include: reliable communication mechanisms, cryptographic support, and data and resource protection functions.

[SC0,SC1]

ACCEPTANCE CRITERIA

OBJ_DES_010

One or more security functions shall be identified for each security requirement. The effectiveness of each security function to fulfill the security requirements shall be documented. The acceptance criteria shall also take into account the following points:

- the effective deployment of the security functions;

[SC0,SC1]

- the documentation of the SECURITY functions. A general categorization can be used, including for example:
  - SECURITY auditing and SECURITY management;
  - communication;
  - cryptography, including identification and authentication;
  - data protection and data privacy;
  - SECURITY function protection, including hardware protections;
  - clock management;
  - resource usage.

[SC0,SC1]

OBJECTIVES

OBJ_DES_020

Use and deploy adequate COTS.

In particular the SOFTWARE COMPONENT shall:

- use well-identified and robust COTS libraries. Such COTS shall be used according to their specified APIs;
  [SC0,SC1]
- use COTS libraries for SECURITY functions based on cryptography;
  [SC0,SC1]
- sanitize data and calls from/to external dependencies with low-confidence level, such as proprietary drivers.
  [SC1]

OBJ_DES_030

Use coding rules for the development of the SOFTWARE COMPONENTS.
[SC0,SC1]

ACCEPTANCE CRITERIA

OBJ_DES_020

The acceptance criteria shall also take into account the following points:

- when relevant, the existing track record (deployment scale) of COTS implementing the SECURITY functions will be examined. COTS shall comply with the Checking objectives OBJ_CHE_010 and OBJ_CHE_020;
  [SC0,SC1]
- when used as a high-criticality SOFTWARE COMPONENT, the SOFTWARE SYSTEM shall be checked for inconsistent use of its interfaces.
  [SC1]

OBJ_DES_030

The acceptance criteria shall also take into account the following points:

- dangerous constructions, such as the one from NIST or MISRA recommendations,shall be detected;
  [SC0]
- the acceptance criterion shall examine the absence of dangerous constructions by the use of tools;
  [SC1]
- code complexity shall be measured with appropriate indicators, for example:
  • function size;
  • number of conditional branches;
  • size and number of stack adjusts
  and shall be refactored to lowerize the cyclomatic complexity of the code.
  [SC0]

OBJECTIVES

OBJ_DES_040

Deprecate dangerous functions including unmaintained SOFTWARE UNITS and SOFTWARE COMPONENTS.

[SC0,SC1]

ACCEPTANCE CRITERIA

OBJ_DES_040

The acceptance criteria shall also take into account the following points:

- The acceptance criterion shall examine the absence of deprecated functions by using dedicated code analysis tools. Modern compilers shall also be configured to warn against unsafe functions at build-time, or perform automatic substitution with more robust functions. When no such configuration is available, the reasons shall be documented.

  [SC0]

- The acceptance criterion shall examine the absence of deprecated functions by the use of tools. If the confidence in the tools results is not as expected, a manual review shall be performed.

  [SC1]

- Unsafe but non-deprecated functions shall also be checked for and, whenever possible, replaced by better-bounded functions.

  [SC1]

OBJECTIVES

OBJ_DES_050

Incorporate the use of application defense mechanisms such as compiler SECURITY options, link-time armoring and load-time defenses.

[SC0,SC1]

Any activity described in these guidelines may be achieved either manually or automatically using SOFTWARE SUPPORT TOOLS, usually chose and qualified during the software design phase.

ACCEPTANCE CRITERIA

OBJ_DES_050

The acceptance criterion shall examine, when applicable, the presence of:

- compile-time defenses such as stack guards, function fortification and control-flow integrity mechanisms;
  [SC0,SC1]
- link-time defenses such as ASLR (Address Space Layout Randomization);
  [SC0,SC1]
- load-time defenses such as code signing and verification.
  [SC1]

Manual Means

ADVANTAGES

The mastering of the activity is ensured when carried out by an expert vanced SECURITY requirements. It will require a bigger effort, but will result in more confidence towards the SECURITY of your software.

DISDVANTAGES

- Tedious activity that may degrade the quality of the outputs
- Completeness (no forgotten case) is difficult to demonstrate
- *Software checking* results are not immediate

Automatic Means

ADVANTAGES

- Repeatability
- Completeness
- Systematization
- Immediacy

DISDVANTAGES

- The tool management shall be controlled by an expert
- Human disengagement
- A qualification shall be performed to provide confidence in the tool outputs

The usefulness of manual and automatic means shall be analysed for each activity to ensure that it is correctly carried on.

OBJECTIVES

OBJ_DES_060

Identify each SOFTWARE SUPPORT TOOL used at any step of the software system development.
[SC0,SC1]

OBJ_DES_070

Operate the SOFTWARE SUPPORT TOOL within its usage domain (described throughout its user manual) taking into account the possible limitations or requirements described in its safety manual (and/or SECURITY manual).
[SC0,SC1]

OBJ_DES_080

Perform a cybersecurity oriented analysis on each SOFTWARE SUPPORT TOOL to ensure that it does not have an impact on the confidentiality, integrity & availability of the SOFTWARE SYSTEM.
[SC1]

ACCEPTANCE CRITERIA

OBJ_DES_060

Identify each SOFTWARE SUPPORT TOOL The acceptance criteria shall document:

- the name;
- the version;
- the purpose of the tool;
- the phase of the development when the tool is used.

[SC0,SC1]

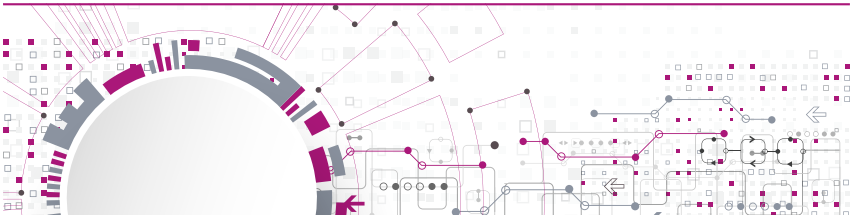
OBJ_DES_070

The acceptance criteria shall document how and when the tools are used. Those criteria include configuration used for each tool (e.g.parameters, scope of the tool) and how they interact with the SOFTWARE SYSTEM or the SOFTWARE COMPONENTS. This includes, when existing, the verification of all exported requirements given in the safety/SECURITY manual.
[SC0,SC1]

OBJ_DES_080

The analysis shall include a description of the interactions between the tools and the SOFTWARE SYSTEM (e.g. possible code modifications introduced by the tool).
[SC1]

2.3 Checking: scans and analyses



OBJECTIVES

OBJ_CHE_010

Check the code for common weaknesses.

The code of SOFTWARE COMPONENTS shall be analyzed for common weaknesses as documented by, e.g. CWE databases, and screened for relevance to the software system. SOFTWARE SUPPORT TOOLS for code verification, through testing or static analysis, shall be used.

[SC1]

ACCEPTANCE CRITERIA

OBJ_CHE_010

The acceptance criteria for the absence of common weaknesses shall examine:

- the scope of weaknesses verified, in particular in relation to the programming language and the used libraries. Beyond intrinsic program errors (memory handling, object manipulations), special attention shall be paid to control-flow and data-flow vulnerabilities;
  [SC1]
- the scope of the checks, especially in relation to the SECURITY perimeters. Coverage information in the case of testing, and reachability information for static analysis, will be examined;
  [SC1]
- the use of formal verification tools to check the absence of code weaknesses is considered highly efficient, in particular for critical logic components and SECURITY functions.
  [SC1]

OBJECTIVES

OBJ_CHE_020

Perform SECURITY testing.

The SOFTWARE COMPONENT shall undergo tests at SYSTEM-level. Tests shall perform extensive explorations of behaviors of the TARGET OF EVALUATION.

[SC0,SC1]

ACCEPTANCE CRITERIA

OBJ_CHE_020

The acceptance criterion for SECURITY testing shall examine:

- the test environment (hardware, network) and its distance to the TARGET OF EVALUATION;
- the volume of test scenarios;
- the coverage of tests, in particular for interfaces and SECURITY functions. Coverage criteria shall include the resistance to malformed and unsanitized inputs;
- the use of binary analysis techniques for test case generation is considered highly efficient.

[SC0,SC1]

2.4 Operations : monitoring and evolutions

OBJECTIVES

OBJ_OPE_010

Monitor the SOFTWARE COMPONENTS for dangerous behaviors and implement safe fall-back options. These may include, depending on the SOFTWARE CATEGORY, graceful termination, logging, or degraded operation strategies.

[SC1]

ACCEPTANCE CRITERIA

OBJ_OPE_010

The acceptance criterion for the monitoring of the SOFTWARE SYSTEM shall examine:

- the amount of information available for e.g. forensics purposes;
  [SC1]
- the scope of behaviors monitored, in particular in relation to the programming language and the used libraries. Special attention shall be paid to control-flow and data-flow requirements;
  [SC1]
- the safety and SECURITY of SOFTWARE UNITS that implement fallback mechanisms, in relation to the SOFTWARE CATEGORY of the SOFTWARE COMPONENT;
  [SC1]
- the relationship between checked behaviors, monitored behaviors, fallback mechanisms, and analyzed weaknesses.
  [SC1]

OBJECTIVES

OBJ_OPE_020

Management mecanisms for SECURITY updates shall be implemented, and processes for updates shall be put in place.

- Updates shall not compromise the safety & performance of the COMPUTER-BASED SYSTEM, and special attention will be paid to modifications of interfaces between SOFTWARE UNITS and/or SOFTWARE COMPONENTS.
- Authentication processes and integrity checking processes shall be part of the update mechanisms.

[SC0,SC1]

OBJ_OPE_030

Manage SECURITY operations. Procedures shall be in place to address:

- active monitoring of threats on SOURCE CODE (including COTS);
  [SC0,SC1]
- active monitoring of threats on SOFTWARE SUPPORT TOOLS;
  [SC1]
- maintenance activity planning, including authorizations and checklist of actions;
  [SC0,SC1]
- backup policy in case of attacks & corruption;
  [SC0,SC1]

ACCEPTANCE CRITERIA

OBJ_OPE_020

- Acceptance mandates that interface compatibility is documented.
  [SC0,SC1]
- When updating a SOFTWARE SYSTEM or a SOFTWARE COMPONENT,regression testing shall be used as an additional acceptance criteria.
  [SC1]
- The integrity (by a checksum or a CRC) of the update file shall be verified before the installation.
  [SC1]
- Authentication mechanisms shall be used before performing updates, to ensure their authenticity (e.g. digital signature).
  [SC0,SC1]

OBJ_OPE_030

The acceptance criterion for management procedures shall examine:

- a monitoring to ensure that the SOFTWARE SYSTEM is up to date regarding the state of the art in security. It includes a monitoring of weaknesses (CWE) and publicly-known vulnerabilities (CVE) from CERT databases. It includes as well field monitoring by controlling logs;
  [SC0,SC1]
- a monitoring to ensure that used SOFTWARE SUPPORT TOOLS are up to date regarding the known vulnerabilities;
  [SC1]

OBJECTIVES

- logs of updates and incidents, based on SECURITY functions for auditing purpose.
  [SC1]

OBJ_OPE_040

Remove sensitive information contained in the SOFTWARE SYSTEM for the decommission.
[SC1]

ACCEPTANCE CRITERIA

- the documentation of the procedures, and of its evolutions. This can include automatically-generated documentation (e.g. automatic update logs);
  [SC0,SC1]
- all tools and data used to develop the SOFTWARE SYSTEM shall be used under version control and shall be registered to allow a complete recovery;
  [SC0,SC1]
- the qualification of operational staff, with a focus on education and training.
  [SC0,SC1]

OBJ_OPE_040

The acceptance criterion for the decommission shall include:

- the physical erasing of all credentials and secrets of the SOFTWARE SYSTEM;
  [SC1]
- the deletion of all the sensitive documentation.
  [SC1]

APPENDIX

Appendix 1 -
Security assessment activity and associated checklist

OBJECTIVE	SC 0	SC I	VERIFIED	APPLIED
2 SOFTWARE CATEGORY				
OBJ_SYS_010	X	X		
OBJ_SYS_020	X	X		
OBJ_SYS_030	X	X		
OBJ_DES_010	X	X		
OBJ_DES_020	X	X		
OBJ_DES_030	X	X		
OBJ_DES_040	X	X		
OBJ_DES_050	X	X		
OBJ_DES_060	X	X		
OBJ_DES_070	X	X		
OBJ_DES_080		X		
OBJ_CHE_010		X		
OBJ_CHE_020	X	X		
OBJ_OPE_010		X		
OBJ_OPE_020	X	X		
OBJ_OPE_030	X	X		
OBJ_OPE_040		X		

Table 2: SECURITY Assessment Checklist

Appendix 2 -
Example of threats identification & classification

In this appendix, examples for the perimeter identification and the threat analysis objectives are given. Those examples provide an overview of what to expect as outputs.

Basically, a perimeter identification for a SOFTWARE COMPONENT or a SOFTWARE SYSTEM consists of identifying every possible inputs or outputs. It is easier to do it graphically with data flow diagrams, even for large or complex systems :

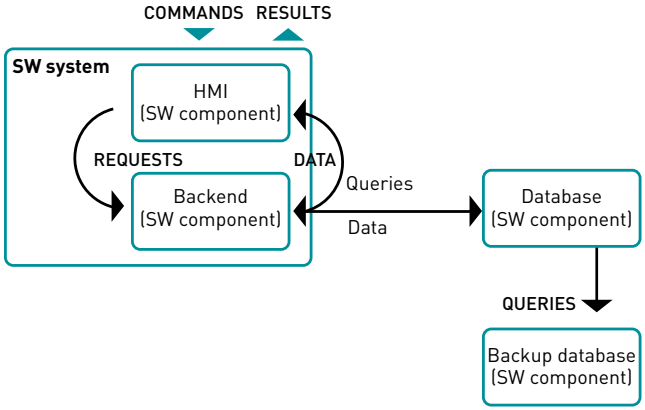


Figure 3: Perimeter identification example.

From this starting point, it is possible to determine what is called «trust boundaries».

They represent (imaginary) crossing lines where the level of trust in the data you handle changes, either between the software components, or between one of the component and an external one (e.g. remote database accessible through Internet, WebService). In this example, it is safer to check that the data sent or retrieved from the database is properly sanitized, to avoid SQL injections or to avoid retrieving corrupt data to the backend. This represents the database boundary.

It shall also be checked if the program files are not corrupted or tampered with when launching the software, for example by checking the DLL signatures or the scheme of the configuration files. This is the storage boundary.

If the system interacts with users through an HMI or an API, it shall be checked that the data received is on par with the specifications or the expected values (e.g. type and length, incorrect encoding, injected statements). This is the system boundaries.

For the sack of the example, a remote backup database was also added, located outside the perimeter of the software system. It is used to replicate the main database and is synced through an uplink (e.g. Internet, local network link). Typically, it is not expected to blindly replicate the main database if it gets corrupted, so there shall be additional checks before effectively importing the data. This is symbolised by the uplink boundary.

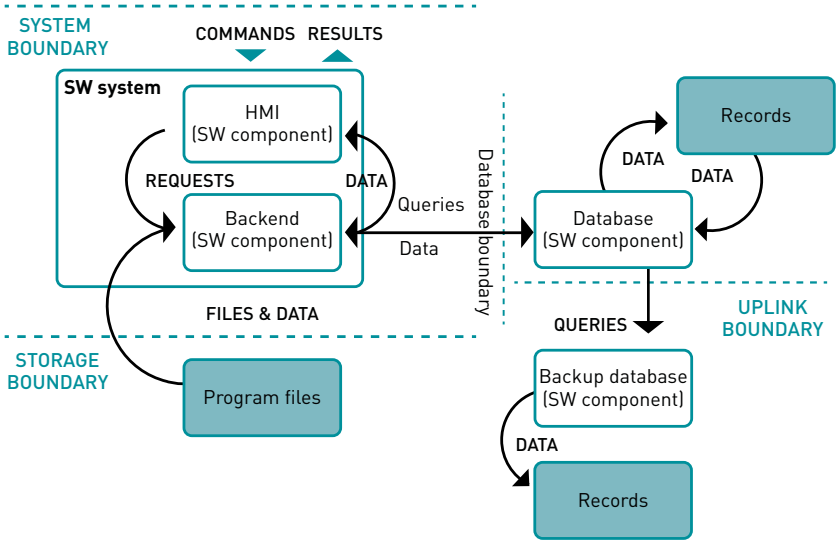


Figure 4: Trust boundaries identification example. (Credits: Bureau Veritas)

To fully complete the security system architecture objectives, it is now needed to perform a threat analysis on each elements from the boundaries identification. It typically starts with elements that violates the trust boundaries (by crossing them), as they are more likely to generate security situations. There are a lot of methodologies or classifications that can be used to classify potential threats,- such as OCTAVE or STRIDE.

You can find an example of STRIDE applied to the previous system below (only the critical links are shown):

Boundary	Link	S	T	R	I	D	E
System	User -->HMI	X		X		X	X
System	HMI--> User		X		X		
Database	Backend --> Database		X		X	X	
Database	Database --> Backend		X		X	X	
Uplink	Database--> Backup		X		X	X	
Storage	Program files --> Backend		X				

Table 3: STRIDE categorization example. (Credits: Bureau Veritas)

As a reminder :

- S: Spoofing
- T: Tampering
- R: Repudiation
- I: Information disclosure
- D: Denial of service
- E: Elevation of privileges

Now that the threats against each critical data flows have been identified, it is possible to derive security requirements for each of them. For example, to prevent the tampering of the program files (last row of the table), an integrity check must be performed (e.g. digitally sign the files, or perform a CRC check at launch). The technical requirement associated ot this security requirement can be refined later in the development process (cf. OBJ_DES_010).

Appendix 3 -
References & existing cybersecurity frameworks

References

- Guidelines for Development & Assessment of Software, Bureau Veritas, BV-SW-100, 2016
- IEC 15408, Information Technology - Security techniques - Evaluation criteria for IT security, IEC, 2009
- IEC 62443, Security for Industrial Automation and Constrol Systems, IEC, 2009
- IEC 61508, Functional safety of electrical/electronic/programmable electronic-safety-related systems, IEC, 2010
- Microsoft SDL: http://www.microsoft.com/en-us/sdl/default.aspx
- OCTAVE: https://www.cert.org/resilience/products-services/octave/
- STRIDE: https://en.wikipedia.org/wiki/STRIDE_%28security%29
- MITRE CWE: https://cwe.mitre.org/
- MITRE CVE: https://cve.mitre.org/

Existing security standards and frameworks

Common Criteria ~ An international standard for establishing the SECURITY properties of a digital system. It relies on quality assurance processes to ensure that the design, development and validation of the system reaches given levels of SECURITY. The standard is referenced as ISO/IEC 15408.

ISO 27xxx Series of standards to deal with Information SECURITY Management System.

IEC 62443 ~ An international standard composed of many volumes to deal with-SECURITY of Industrial Automation and Control Systems. This is aimed to be the reference in the industry based on a top-down approach (Part 1: General, Part 2: Policy & Procedure, Part 3: System, Part 4: Component).

NIS Directive ~ The «Directive on Security of Network and Information Systems» is an European legislation that focuses on network and infrastructure SECURITY. The Directive mandates the use of risk management practices and systematic incident reporting for certain digital companies.

Guideline on Cybersecurity Onboard Ships ~ A set of recommendations by actors of the Marine and Offshore industry, led by the BIMCO shipping association. The aim is to build SECURITY awareness through a cycle of procedures. It sets high-level objectives that include onboard software systems assessments.

IEC 61162 ~ An international standard composed of many volumes to deal with digital interfaces (radio communication) for navigational equipment within a ship.

NIST Cyber framework ~ A set of recommendations of the National Institute of Standards and Technology (from the US Department of Commerce) on SECURITY. It gathers referentials, good practices and methodology to handle SECURITY.

SDL ~ The Security Development Lifecycle was developed and is currently maintained by Microsoft. It intends to bring some SECURITY procedures and requirements in the software development cycle. For each step of this cycle, the SDL documentation explains what kind of SECURITY checks can be performed, and what kind of tools can be used to do so.

RGS ~ A set of requirements issued by the French administration regarding SECURITY. These requirements are to be followed when a private company has to operate or interconnect itself with a governmental body, and are thus believed to be state-of-the-art documentations, especially regarding cryptography.

CSPN ~ A certification by the French National Cybersecurity Agency (ANSSI), delivered upon successful evaluation of the SECURITY of a digital system. The evaluation is performed by a licensed service provider.

These technical guidelines were developed thanks to the experts from the List, an institute of CEA Tech, and the Bureau Veritas Dependability team.

© - Copyright Bureau Veritas SA, Photos: Bureau Veritas, Shutterstock - All rights reserved | BV-SW 200/20170609

Move Forward with Confidence

Bureau Veritas SA
Société Anonyme – RCS registration number: 775 690 621 R.C.S Nanterre
Head office address: Immeuble Newtime
40/52 Boulevard du Parc – 92200 Neuilly-sur-Seine
software@fr.bureauveritas.com
www.bureauveritas.com