



DEBIOTECH

How to:

Handle

cybersecurity

for your

medical device?

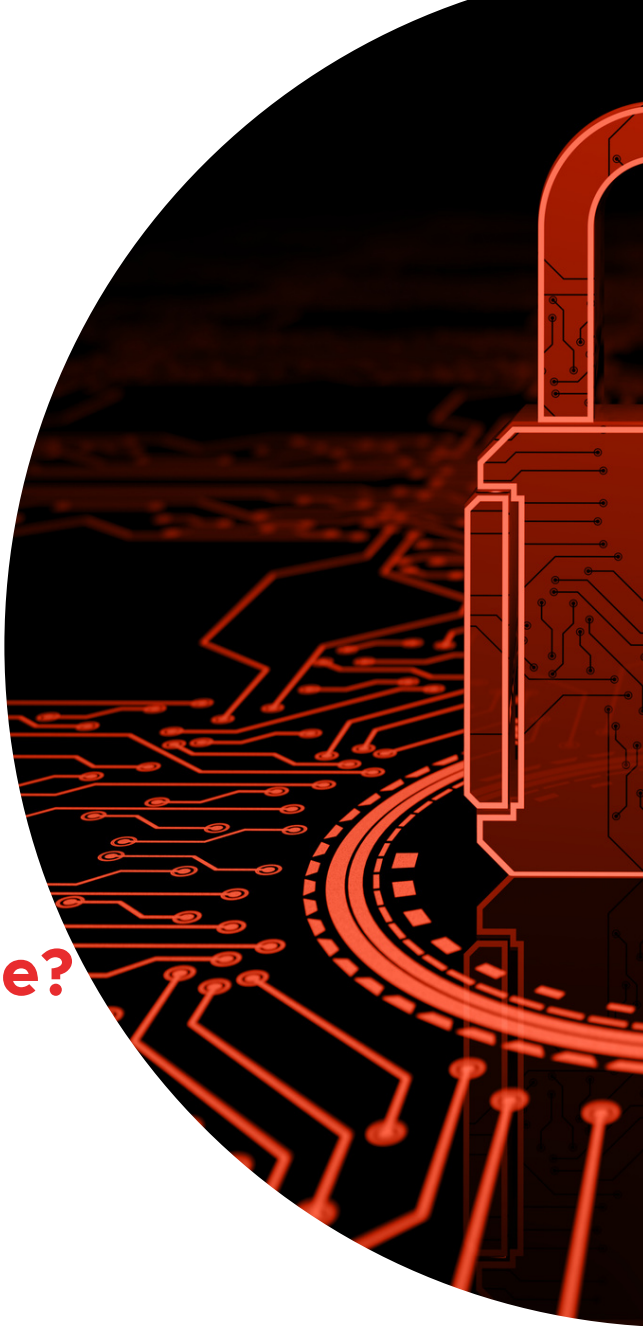


Table of content

1/ Goal of this publication	3
2/ Introduction	3
3/ Safety vs. Security	4
4/ Good practices before applying security risk management	5
4.1/ Controlled use of third-party components (SOUP, OTS, OS)	5
4.2/ Smart use of cryptography	6
4.3/ Confidential computing	6
4.4/ Remove compilation warnings	6
4.5/ Use of static code analyzer	7
4.6/ Never trust external inputs	7
4.7/ Use of hardware features	8
4.8/ Pay attention to memory management	8
4.9/ Identify & protect sensitive data	8
5/ Software lifecycle security process	9
5.1/ Write/update software security management plan	11
5.2/ Identify security risks	12
5.2.1/ Scope of your device: data & communication functionalities	12
5.2.2/ Scan for Common Weakness Enumeration (CWE)	13
5.2.3/ Common Vulnerabilities and Exposures (CVE) identification	14
5.2.4/ Threat modeling	15
5.2.5/ Post-market security risk identification	16
5.3/ Evaluate security risk levels	17
5.3.1/ Software security scoring scheme	17
5.3.2/ Quantify security risk levels associated with identified threats	18
5.4/ Identify security risk control measures	18
5.5/ Implement security risk control measures	19
5.6/ Verify efficacy & effectiveness of risk control measures	19
5.6.1/ Common vulnerability testing	19
5.6.2/ Malware testing	19
5.6.3/ Malformed input testing	20
5.6.4/ Penetration testing	20
5.6.5/ Security risk management report	21
5.7/ Release & distribute update	21
5.8/ Market withdrawal & decommissioning	22
6/ Regulatory landscape	22
7/ Authors	23
8/ Next steps	23

1. Goal of this publication

is to support you during the development of your device with a focus on a very hot topic: **Cybersecurity**.

With this content, we aim to help you better understanding how Cybersecurity (also called software security) can impact your process of development and the design of your device.

2. Introduction

With the increasing number of connected services and the advent of IoT era, the risk of cybersecurity attacks is higher than ever. Medical devices and services are not spared by this threat and consequences of an attack on your device can be disastrous:

- ⚠ Risks for patients 'safety
- ⚠ Risk of patient data theft
- ⚠ Recall of your product with dramatic consequences for your company's reputation and return merchandise authorization costs.

After being neglected for a long time, Cybersecurity is now a critical part of every project. When starting a new project, you must now keep in mind that Cybersecurity is not a feature that you can add at the end, it must be considered at a very early stage of your product design, integrated into your development process and followed-up once your device is on the market.

3. Safety vs. security

It is important to distinguish safety from security concepts.

On one hand, medical devices must ensure patient safety while ensuring specific performances. Safety risk control measures shall be implemented to reduce safety risks to an acceptable level for users and patients and be as robust as possible against failures that may lead to harm of various severities. Safety focuses on protection of users and patients' health.

On the other hand, medical devices must ensure software security. Security risk control measures shall be implemented to reduce security risks associated with malicious use of the device features. Security focuses on protection of data and software systems.

Therefore, if you do not treat those two aspects properly you may develop safe but unsecure devices or secure but unsafe devices.

In the worst case, an insufficient security level can lead to a major safety risk (unexpected remote access to a life supporting device, upload of an unsafe treatment, etc.), which may create critical effects on the patient (inappropriate treatment received, death, etc.). Strong safety and security risk control measures shall be implemented to make this situation impossible or highly improbable.

Debiotech recommends to:

- Make the distinction between safety and security in your risk management processes and files.
- Implement strong safety risk control measures and verify their efficacy and effectiveness. As much as reasonably possible use non software risk control measures to avoid introducing software components with high-risk classification.
- Implement strong security risk control measures and verify their efficacy and effectiveness.



Safety



Security

4. Good practices before applying security risk management

To minimize your development work when applying a security risk management process, you need to start minimizing software weaknesses at the beginning of your development. This can be achieved by reducing as much as possible the attack surface of your system. In this chapter multiple concepts allowing you to start early on with good software security practices will be developed. Most of those concepts impact your software and hardware architecture and are therefore important to consider in the early stages of your development.

4.1. Controlled use of third-party components (SOUP, OTS, OS)

Operating Systems (OS), Software of Unknown Pedigree (SOUP), Off-The-Shelf (OTS) components and your own code must only use strictly necessary features. The more protocols, drivers or libraries are used, the higher the risk of introducing vulnerabilities.

A quick look at the Common Vulnerabilities and Exposures databases searching for “Linux Kernel” will return many CVEs related to specific drivers that are not necessarily required on your device. This demonstrates that you should limit the use of OS drivers to the strict minimum and block or deactivate unnecessary OS features from your system.

Nowadays, many very useful libraries are available; however, very few of them have been developed following a development process compliant with medical regulations. It is not forbidden to use them, but you should carefully consider the amount of work that will be needed to integrate such libraries in your system. To avoid this additional and complex work of verifying and validating external libraries, you should strongly limit their use and only use them when it is relevant for performance and robustness objectives. These components must be isolated as much as possible from safety-critical components.

Debiotech recommends to:

- Limit as much as possible the use of SOUP
- Limit as much as possible the use of protocols and drivers associated with the operating systems of your system,
- Deactivate software communication features that you do not need.

4.2. Smart use of cryptography

Most of the connected devices need to use strong cryptography algorithms to secure authentication, data storage, data communication and sometimes remote computing. Cryptography is a highly expert field requiring very specific skills and expertise. It is rarely the core of your team expertise, and you should just accept it. Do not try to write your own cryptography algorithms; use state-of-the-art available solutions instead. Well-tested solutions exist for all languages. In embedded systems, “openssl” for Linux devices or “mbedTLS” for bare-metal devices are good choices that will provide all needed features.

Be careful, using a bug-free library does not mean that bug-free cryptography will be written. The documentation must be followed to use the functions as they are intended to and for the algorithm to be correctly used (e.g., a common mistake is to use a fixed initialization vector for AES encryption).

Finally, be aware that algorithms may become obsolete because a flaw is discovered, or because it becomes too weak in comparison to the increase of computational power. Only algorithms that are considered safe for the next coming years must be used.

[The FIPS 140-2 Algorithm Lists](#) can be checked to know which algorithms are approved.

4.3. Confidential computing

If you are using remote computing, you should consider using confidential computing for your remote computing server. Data encryption for storage and transmission is now relatively popular. However remote or cloud-based computing is usually decrypting data before processing it, so data can be accessible and readable at the processor level. This can make your data accessible by attackers or by the provider of your remote server infrastructure. New secure computing technology exists (as for example the ones provided by: IBM, Microsoft Azure, Google, Intel and Cysec) and it allows to render the entire data chain confidential: from storage to computing, including transmission.

4.4. Remove compilation warnings

When developing your software, you might have the tendency to ignore warnings in your build summary. They are the sign of weaknesses of your code that might not directly impact the performance of your device but might reduce its robustness and security.

Debiotech recommends to

- fix warnings systematically and quickly
- use compilation options that protect against stack overflow and certain Return-oriented programming (ROP) attacks.

4.5. Use of static code analyzer

It is recommended (UL-2900-2) to make use of static code analysis to reduce software weaknesses like, for instance, the use of predictable random number seed, the use of buffer above its limit, unmanaged processor exceptions, allow the use of corrupted code, or allow the unexpected use of sensitive data (patient data, treatment data or encryption keys).

4.6. Never trust external inputs

Every data external to the application must be checked before being processed. This includes, but is not limited to, data files stored on disk, user input and data coming from external communication channels (Wi-Fi, Bluetooth, ...). For the developers, it means that special care must be taken while handling this data:

- ✓ Input files must be authenticated using cryptographic signatures,
- ✓ User input must be checked. Some basic principles must be followed:
 - Check data length before copying it into buffers to prevent buffer/stack overflows,
 - Use regex if a specific pattern for input is expected,
 - Sanitize your inputs to prevent SQL injection.
- ✓ Always authenticate the peer before exchanging data. Secure protocols must always be used (e.g., Transport Layer Security (TLS) 1.2 at least for network communications)
- ✓ All the code impacted by external input must be identified and tested.

```

american fuzzy lop 2.57b (fuzzgoat)

process timing
  run time      : 0 days, 0 hrs, 30 min, 41 sec
  last new path : 0 days, 0 hrs, 0 min, 31 sec
  last uniq crash : 0 days, 0 hrs, 3 min, 40 sec
  last uniq hang : none seen yet
cycle progress
  now processing : 465 (97.28%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : interest 32/8
  stage execs : 28.4k/34.5k (82.27%)
  total execs : 5.35M
  exec speed  : 2628/sec
fuzzing strategy yields
  bit flips : 28/116k, 16/116k, 9/115k
  byte flips : 0/14.6k, 0/13.4k, 1/12.8k
  arithmetics : 82/768k, 0/176k, 0/6650
  known ints  : 3/69.9k, 0/367k, 1/526k
  dictionary  : 0/0, 0/0, 0/0
               havoc : 370/3.01M, 0/0
               trim  : 77.38%/6076, 5.15%

overall results
  cycles done : 9
  total paths : 478
  uniq crashes : 33
  uniq hangs  : 0

map coverage
  map density : 0.30% / 0.92%
  count coverage : 3.64 bits/tuple

findings in depth
  favored paths : 114 (23.85%)
  new edges on : 164 (34.31%)
  total crashes : 4219 (33 unique)
  total tmouts : 31 (11 unique)

path geometry
  levels : 15
  pending : 116
  pend fav : 5
  own finds : 477
  imported : n/a
  stability : 100.00%

[cpu000: 92%]

*** Testing aborted by user ***

```



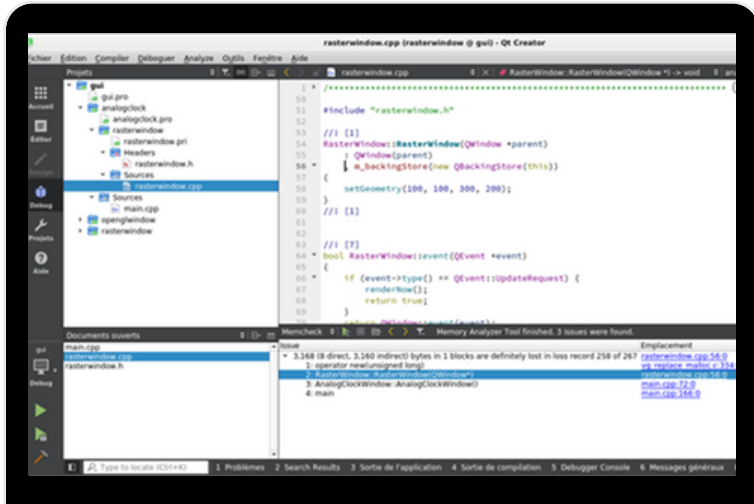
Fuzzing tools like American Fuzzy Lop (AFL) may help in this process.

4.7. Use of hardware security features

Where possible, use hardware encryption capability on the communication port and specific and secure storage hardware components is recommended.

4.8. Pay attention to memory management

Memory management is critical in many applications. Programs that allow dynamic memory allocation without protection may be dangerous as they can lead to errors such as information disclosure or arbitrary code execution if not handled properly. Developers must be aware of the problems related to memory leak, double free, stack management, or dangling pointer dereference. **Special care** must be taken when using dynamic memory. Prefer usage of smart pointers and test the code to detect memory-related errors.



Instrumentation framework like Valgrind may help.

4.9. Identify and protect sensitive data

Medical devices and most embedded systems store their data in embedded MMC (eMMC), or other type of flash memories, in an unencrypted form. Keep in mind that even if the memory is soldered to the device, it is easy to read/write/modify its content by soldering a couple of wires. It is often not feasible to encrypt the entire disk for performance reason, but all personal health information (PHI) and secrets (e.g., tokens used to access web APIs) must be encrypted. The cryptographic keys must be stored securely in a dedicated area of the microcontroller if it provides anti-tamper protection, or in a Secure Element (SE). A SE highly improves the device security by providing safe storage, secure key generation and more. Another alternative is to strictly limit the PHI stored on the device.



eMMC memory

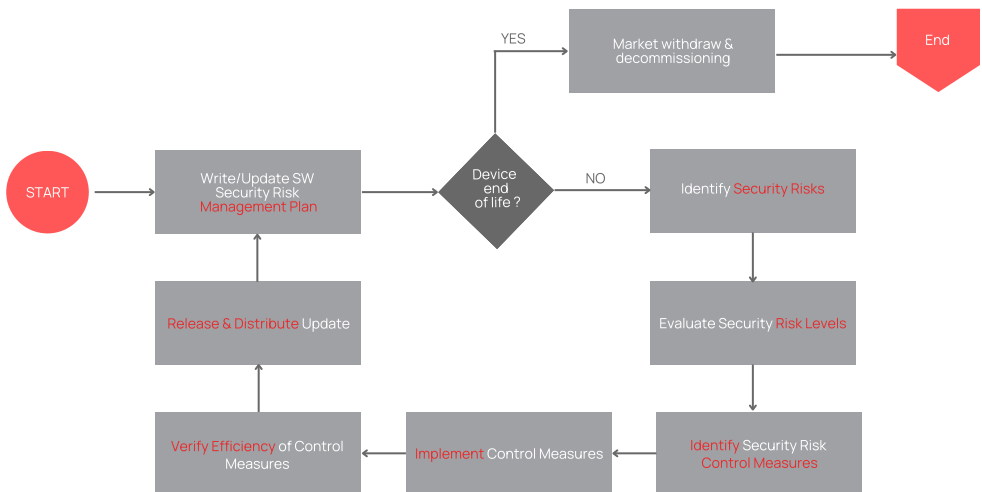


A SE memory

5. Software lifecycle security process

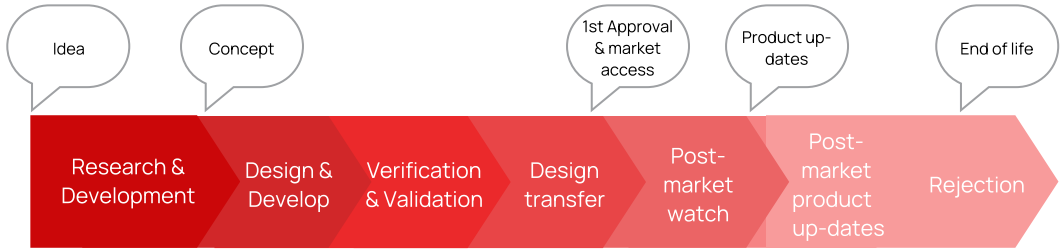
Software security must be under control from the beginning of the realization of the product to its end of life. It must consider the developed components as well as the integrated third-party components and their environment of use.

Software security management activities must be applied at all stages of the development and distribution of your medical device. A procedure for software security management must be developed and a software security management plan must be established for each project. Activities involved in this procedure are illustrated in the next figure and will be used to structure the content of this publication.



For each of these activities, the work to be executed depends on the phase of your project. As a reminder, you'll find below the typical phases of the development and marketing of a medical device.

In parallel of the application of below flowchart you will have to keep up to date the list of files you generate to document your software security management activities. This is usually done within the Design History File of your project. A method, usually well appreciated by reviewers is to use tags within your Design History File to quickly identify files associated with your software security management activities.



In addition, at each transition from one phase to another, you must document the files and activities you carried on in the previous phase and demonstrate the completeness of your work and the non-ambiguity and consistency of the content elaborated in the different files. This is usually performed through gate or phase reviews.

Debiotech recommends to:

- Describe your procedure for software security risk management, early in your development process.
- Refer your software security procedure in your design and development procedure and describe for each phase of your D&D procedure the applicable activities of the software security procedure.
- Create the software security risk management plan for your project, based on the software security risk management procedure.
- Document the activities and the files related to software security risk management, using gate or phase reviews of your D&D procedure.
- Keep up to date your Design History File with the files related to software security risk management.

5.1. Write/update software security risk management plan

Your activities for security risk management must be clearly identified and integrated within your project plan. You can have a specific document to describe your security risk management plan or integrate it into the project plan. This document allows you to illustrate your planned activities, to document assigned resources and expected timing and finally to demonstrate the completeness of your approach in the handling of security risk management aspects. Your plan is a living document that must be kept up to date. Change of plans are normal, especially in early-stage projects but lack of, or clearly inaccurate plans, will be challenged by reviewers. The Figure 8 illustrates typical activities to be included within your security risk management plan.

Debiotech recommends to:

- Write down your Security risk management plan at the beginning of the project even if you lack visibility on its mandatory content.
- Keep it **simple** and high level. Details can come with other documents. The importance is to document the different activities you will have to perform at the different phases of your project, to establish a timeline and assign resources.
- Ensure existence and use of a security risk management plan (or integrated to your project plan) as a complement to your Design & Development procedure. However, the filling of this template to generate a plan specific to your project is necessary.
- Keep it up to date to ensure your plan is realistic.

ACTIVITY	DUE DATE	ASSIGNED COLLABORATORS	REQUIRED HW&SW MATERIALS
Identify Security Risks	15.06.2025	John Doe - Cybersecurity Expert Jane Smith - Project Manager Diane Cruise - SW Developer	SAST Tool XXX MS Threat Modeling Tool
Evaluate Security Risk Levels	15.06.2025	John Doe - Cybersecurity Expert Jane Smith - Project Manager Diane Cruise - SW Developer	CVSS Calculator
Identify Security Risk Control Measures	25.06.2025	John Doe - Cybersecurity Expert Jane Smith - Project Manager Diane Cruise - SW Developer	MS Threat Modeling Tool
Implement Risk Control Measures	15.08.2025	Diane Cruise - SW Developer	SW development environment
Verify effectiveness of Risk Control Measures	15.10.2025	Jessy Jones - SW Tester	HW in the loop testing Automatic SW testing tool
Release & Distribute Update	11.12.2025	Jane Smith - Project Manager	Source code management platform

Example of SW security risk management plan

5.2. Identify security risks

5.2.1. Scope your device: data & communication functionalities

To identify the security risks associated with your device you need to establish early in the design and development process your device communication characteristics:

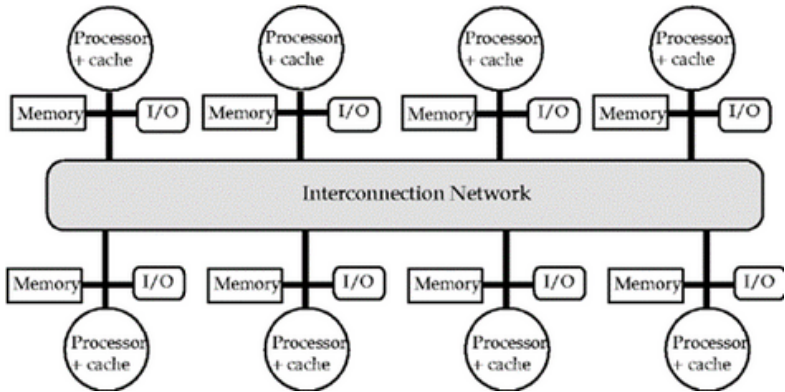
- ✓ The data you store and their location in your system,
- ✓ The communication channels you use, their protocols, the data and the command they send or receive.

This will allow you to clearly identify the expected capabilities of your device and then to identify the associated threats and vulnerabilities:

- ✓ Pure standalone device. Not connected to any server and exchanging data via standard data storage hardware such as USB sticks or CD-ROM.
- ✓ Device within safe network infrastructure. Your device is only connected to the local servers of the clinical center that uses it and can exchange data internally, for example via the use of Pictures Archiving & Communication Systems.
- ✓ Device with possible updates from remote servers. The only use of the connection to internet and other servers is for update of the device.
- ✓ Device with remote monitoring and maintenance from remote servers. Your system sends non-private data to an external server to perform system monitoring and maintenance.
- ✓ Device transferring data to an external server to generate an encrypted centralized database with similar devices.
- ✓ Device transferring data to an external server and using this server to perform computation and receive results from those computations.
- ✓ Devices that can receive new therapies/prescriptions from a health care professional via a remote server.

Debiotech recommends to:

- Identify, early in the development process, the data and communication characteristics of your system.
- Minimize software weaknesses.
- Establish software security scoring scheme (Chapter 5.3.1).
- Perform threat modeling (Chapter 5.2.4).



Data, processors and memory location models

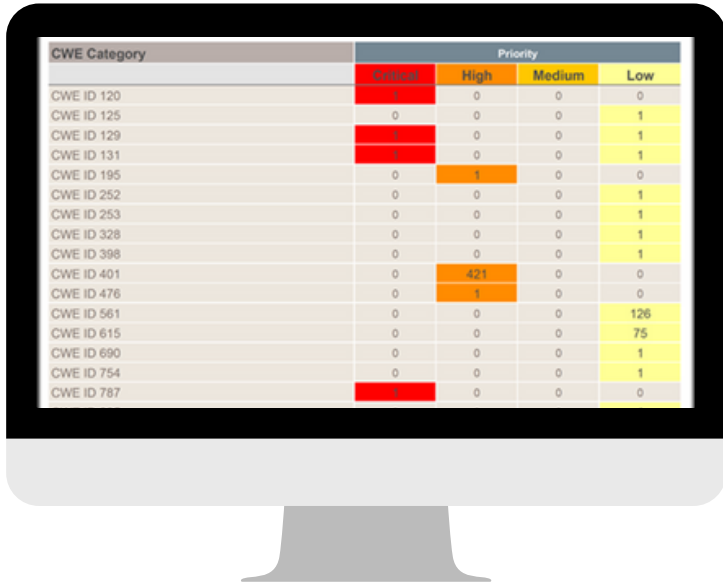
5.2.2. Scan for Common Weakness Enumeration (CWE)

The CWE is a list of common types of weaknesses that are typically found in software. You can think of it as a **dictionary of weaknesses** that are found in software. This list is maintained by MITRE, and it is freely accessible.

The most effective method to identify CWEs in your code is to use tools such as Static Application Security Testing Tools (SAST tools). These tools scan the source code just as a static code analysis tool would, but they identify potential security issues instead of potential software bugs (some tools do both). These tools are a great way to identify and prioritize the CWEs in your code.

SAST tools allowing to perform these tasks include but are not limited to:

- ✓ Perforce - Klocwork for C, C++, C# and Java,
- ✓ Parasoft - Parasoft Test for C, C++, .NET and Java,
- ✓ Micro Focus - Fortify which supports multiple operating systems and languages,
- ✓ A list of existing tools is maintained by the OWASP (Source Code Analysis Tools/ OWASP Foundation)



CWE Category	Priority			
	Critical	High	Medium	Low
CWE ID 120	0	0	0	0
CWE ID 125	0	0	0	1
CWE ID 129	0	0	0	1
CWE ID 131	0	0	0	1
CWE ID 195	0	1	0	0
CWE ID 252	0	0	0	1
CWE ID 253	0	0	0	1
CWE ID 328	0	0	0	1
CWE ID 398	0	0	0	1
CWE ID 401	0	421	0	0
CWE ID 476	0	1	0	0
CWE ID 561	0	0	0	126
CWE ID 615	0	0	0	75
CWE ID 690	0	0	0	1
CWE ID 754	0	0	0	1
CWE ID 787	0	0	0	0


Illustration of Fortify SAST including CWE classification

Debiotech recommends to:

- Early on in your project execution, start using a SAST tool.
- If possible, make the SAST tool part of your continuous integration environment and define criteria to prioritize the work according to the criticality of the identified weaknesses.

5.2.3. Common Vulnerabilities and Exposures (CVE) identification

This activity is of **critical importance** for proper software security risk management activities. It consists in the identification of the Common Vulnerabilities and Exposures associated with the third-party libraries and the operating systems you are using. Most of the libraries and software components you integrate within your system provide this information and make it publicly available. However, this information evolves with time. You need to frequently review these vulnerabilities to ensure that your system is protected against recently discovered vulnerabilities. You can do this review manually, but you will quickly benefit from tools allowing you to do this automatically such as Software Composition Analysis tools (SCA tools).

 SCA tools allow you to perform the following tasks automatically based on your source code or on your binaries:

- ✓ Software Bill Of Materials (SBOM) generation: identification of all the libraries used, their versions and the associated licenses.
- ✓ Cybersecurity Bill Of Materials (CBOM) generation: identification of all the Common Vulnerabilities and Exposures associated to your SBOM as for example buffer overflows, SQL injection flaws, Authentication problems, Access control issues and Insecure use of cryptography.

You can then focus on the review of those vulnerabilities, determine if they apply or not to your system and if you must develop a patch to remove them. Please remember that you must provide rationales to argue why you estimate that a vulnerability is not applicable to your system.

SAST tools allowing to perform these tasks include but are not limited to:

- ✓ WhiteSource
- ✓ Black Duck
- ✓ JFrog Xray
- ✓ A list of existing tools is maintained by the OWASP (Software Composition Analysis Tools | OWASP foundation)

Debiotech recommends to:

- Early on in your project execution, integrate within your software development infrastructure an SCA tool for SBOM and CBOM generation.
- Frequently use this SCA tool even if no change is made to your software to stay up to date with latest identified vulnerabilities.
- Take the necessary time to identify which vulnerabilities are applicable to your software and provide rationales justifying your position for non-applicable vulnerabilities. For applicable ones, describe the patch you will be developing and the way you will verify its efficacy.

5.2.4. Threat modeling

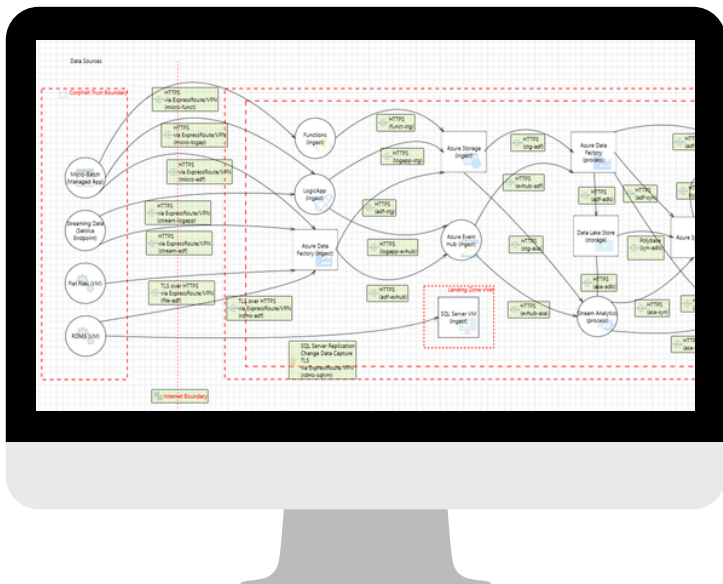
To identify vulnerabilities of your system, you must apply a threat modeling approach. The first step is to list all the elements that will affect the security of your system:

- ✓ Physical ports,
- ✓ Internal and external communications mechanisms,
- ✓ The list of third-party libraries used by the different communication and authentication mechanisms,
- ✓ And more generally all valuable assets of your device.

Then, for each of these elements and assets you need to identify relevant threats. A typical approach for this step consists in using an existing model such as **STRIDE** or alternatives like **LINDDUN** or **CIA Triad** for example. When using STRIDE, for each identified asset, you must evaluate the consequences of:

- ✓ Spoofing: The device/service/user is not the one it is supposed to be, like in Man-in-the-Middle attacks,
- ✓ Tampering: Illegal access to your device, with hardware or software modifications,
- ✓ Repudiation: Claiming to have not performed an action,
- ✓ Information disclosure: Expose sensitive information about your device or private data,
- ✓ Denial of service: The device does not work as expected,
- ✓ Elevation of privilege: Gaining capabilities without authorization.

Finally for each threat, a severity score must be determined to prioritize them. Dedicated tools exist to help with the threat modeling process, like “**Microsoft Threat Modeling tool**” or OWASP “**Threat Dragon**”. These tools will help you to graphically model your system, identify threats using STRIDE and generate a report.



Threat model example using MS Threat Modeling

5.2.5. Post-market security risk identification

Hackers evolve and improve their technics. The list of known vulnerabilities increases with time and shall be reviewed periodically for your device. New critical vulnerabilities might require developing or integrating a new security patch for your software and distribute it to your users. Depending on the security risk level you may have to react quickly, inform your users about the security risk and provide a solution that can be quickly deployed.

5.3. Evaluate security risk levels

5.3.1. Software security scoring scheme

To quantify your security risks and prioritize their handling, you need to apply a clear security scoring scheme.

The Common Vulnerability Scoring System (CVSS), a standard commissioned by US National Infrastructure Advisory Council (NIAC) in 2005 and maintained by the International Forum for Incident Response and Security Teams (FIRST), has been developed and made publicly available as well as an associated calculator allowing you to simply quantify the level of risk associated with the identified vulnerabilities (Common Vulnerability Scoring System Version 3.0 Calculator (first.org)).



Another calculator is provided by NIST and provide a more user-friendly interface and visual graphics.

Another usable scoring system is DREAD, simpler than CVSS but providing less detailed image of the exact nature of the vulnerability and risk.

These scoring systems and calculators can be used to quantify the initial level of risk of each identified vulnerability and its level after mitigation(s). The level of effort you put in minimizing each risk will depend on its initial score. To do so, you shall define rules as illustrated in the next figure, that will give you a systematic approach to the treatment of your identified vulnerabilities.

CVSS Scoring	CVSS Severity Ranking	Definition sand actions
0 - 3,9	LOW	Acceptable level of risk. To be exploited it requires advanced and noticeable social engineering hacking methods or the impact is highly limited.
4,0 - 6,9	MEDIUM	Risk shall be mitigated as reasonably praticable. If the risk is not mitigated further, a rationale shall be provided in the security risk management report.
7,0 - 8,9	HIGH	Not acceptable, must be mitigated as reasonably practical.
9,0 - 10,0	CRITICAL	Not acceptable, must be mitigated. If devices are in the market, must be fixed immediately.

5.3.2. Quantify security risk levels associated with identified threats

Based on the identified threats and the selected software security scoring scheme, you can now quantify the risk level associated to each threat and therefore prioritize their handling. You will therefore obtain threats of different levels and the next steps will depend on their risk level.

Debiotech recommends defining the following rules:

- For medium and low-level threats, you should reduce risk as reasonably practicable while it is mandatory for higher threat levels. If no risk control measure is defined for a given threat a rationale must be provided to explain why this threat cannot be reduced further with reasonably acceptable efforts. In some case, some critical and high threat levels might impact the critical performance of your device and they could be accepted as they are if the ratio benefits/risks remain largely in favor of the patient. In this case a clear and detailed rationale must be provided.

5.4. Identify security risk control measures

To control the security risks associated with your system, you must develop security risk control measures, also often called mitigations. Those mitigations can be of many different types and their impact on the software security risk levels depends on their characteristics.

Usually, **external risk control measures** are highly recommended as they allow to minimize the risks associated to your system without impact it directly. An example of an external risk control measures could be that your system shall be stored in a room only accessible to authorized persons. Those mitigations can be very robust and relatively easy to put in place.

Hardware related risk control measures are also usually highly appreciated for the level of security they bring. For example, to protect sensitive data, you can record them on a secured and a dedicated memory (trust platform), only accessible through a dedicated communication port and using encryption and authentication. Also, secure boot loader is often required for medical devices with a high-risk class.

Typical **software related risks control measures** include the use of strong passwords, double identifications, and use of privileges to manipulate data in the devices. Other usual security risk control measures, more hidden to the users, are the use of encrypted data transmission and the check of the authenticity of the product code. Note that robust encryption technics shall be used. They are listed in NIST FIPS 140-2 Appendix A.

5.5. Implement risk control measures

Once software security risk control measures are identified and their impact evaluated, it is now time to implement them. This can represent a significant amount of work. It is highly recommended to use existing and robust libraries for standard technics as authentication and encryption. However, this might add new libraries and therefore new vulnerabilities to your system. In the case those security risks might be associated with critical safety risks, this will impact the software risk class associated to this library. This can induce a complex situation where the library you used must be compliant with class B or class C software classification according to IEC 62304, which is rarely, if not never, the case.

Debiotech recommends:

- Using non software risk control measures as much as possible to reduce software security or safety risks.
- Clearly separating code dedicated to security feature from those for safety or performance features.

5.6. Verify efficacy & effectiveness of risk control measures

Security risk control measures shall be verified. Their testing shall be part of the software verification plan of the product according to IEC 62304. While some dedicated tests shall be developed for the security risk control measures specific to your product, others are more standard:

5.6.1. Common vulnerability testing

Check that all identified common vulnerabilities associated to the use third party components are handled either through the integration of a patch or through the provision of a clear rationales explaining their non-applicability.

5.6.2. Malware testing

Ensure that no malware is embedded in your software, in the third-party components or in the used operating system using malware screening software.

5.6.3. Malformed input testing

Every check developed to validate external data must be verified. This includes, but is not limited to:

- ✓ Verify proper behavior of input files authentication using cryptographic signatures,
- ✓ Verify that the product is not impacted by malformed inputs (reject data making no sense) which could result in an attempt of hacking,
- ✓ Verify proper behavior of peer authentication before data exchange,
- ✓ Verify proper behavior of code impacted by external input(s).

```

american fuzzy lop 2.57b (fuzzgoat)

process timing
  run time : 0 days, 0 hrs, 30 min, 41 sec
  last new path : 0 days, 0 hrs, 0 min, 31 sec
  last uniq crash : 0 days, 0 hrs, 3 min, 40 sec
  last uniq hang : none seen yet

overall results
  cycles done : 9
  total paths : 478
  uniq crashes : 33
  uniq hangs : 0

cycle progress
  now processing : 465 (97.28%)
  paths timed out : 0 (0.00%)

map coverage
  map density : 0.30% / 0.92%
  count coverage : 3.64 bits/tuple

stage progress
  now trying : interest 32/8
  stage execs : 28.4k/34.5k (82.27%)
  total execs : 5.35M
  exec speed : 2628/sec

findings in depth
  favored paths : 114 (23.85%)
  new edges on : 164 (34.31%)
  total crashes : 4219 (33 unique)
  total timeouts : 31 (11 unique)

fuzzing strategy yields
  bit flips : 28/116k, 16/116k, 9/115k
  byte flips : 0/14.6k, 0/13.4k, 1/12.8k
  arithmetics : 82/768k, 0/176k, 0/6650
  known ints : 3/69.9k, 0/367k, 1/526k
  dictionary : 0/0, 0/0, 0/0
  havoc : 370/3.01M, 0/0
  trim : 77.38%/6076, 5.15%

path geometry
  levels : 15
  pending : 116
  pend fav : 5
  own finds : 477
  imported : n/a
  stability : 100.00%
  
```



Fuzzing tools like American Fuzzy Lop (AFL) may help in this process.

5.6.4. Penetration testing

The best way to verify that all vulnerabilities are correctly fixed is to request a penetration testing lab to confirm that none of them can be exploited.

Structured penetration testing is required by UL 2900-1, section 16. Penetration testing should be run by an independent lab. Such test should be first performed as “Black-box testing”, which means that testers will try to find and exploit vulnerabilities with no (or very little) knowledge of the device. Then, “White-box” testing should be run, where all valuable information (even source code or schematics if necessary) is provided. Also, providing the list of identified CVEs will help the lab confirm that the device does not contain weaknesses.

5.6.5. Security risk management report

A security risk management report shall be generated to summarize the plan followed, the activities performed, and the list of documents generated including a list of the identified threats (from your own device and from third-party components), the associated risk control measures or rationales for their absence, the results of the verification campaign and the instructions for use related to security.

The security risk management plan mentioned in the report must define what are the next expected actions regarding security (improvements and periodic reviews).

It may happen that some of your threats or vulnerabilities verification is not fully successful: this shall be documented within a list of known anomalies. For those remaining anomalies, clear and convincing rationales shall be provided to argue why those anomalies are acceptable for the ongoing software release. If they are not acceptable, those anomalies shall be fixed before releasing the software.

5.7. Release & Distribute Update

Once your system is fully verified and validated, the software version can be released and made available for design transfer and manufacturing. If your system is a standalone software, this means that you can generate your installation files based on this software release. However, the efficacy and effectiveness of the installation files must be validated through the verification of the proper behavior of the software installed using these installation files. This can be done using a sub-set of your verification tests: the installation, operational and performance qualification tests.

Once the proper behavior of the installation files verified, you can finalize your technical documentation for submission to authorities and/or inform your users about the availability of security updates for their installed software. The procedure and tools for such software security update shall be already available to your customers.

Debiotech recommends to:

- Integrate within your system a way to inform your users that a security update is available.
- Integrate within your system a way to easily update their product version with the new one.

5.8. Market withdrawal and decommissioning

The security risk management plan shall also include actions required when decommissioning a device. Those actions shall ensure that no sensitive data remain accessible to possible technical or customer services after decommissioning of the device.

6. Regulatory Landscape

From a regulatory standpoint, data protection and data privacy are usually treated in the same texts. Data security on its side has its own legislations. The applicable regulations usually depend on the type of data: health-related data are usually associated with stronger requirements in term of privacy and security.

Data protection & privacy

- Europe: GDPR,
- Switzerland: Federal Act on Data Protection,
- US: HIPAA and numerous data protection laws enacted on both the federal and state levels.

Data security

- US: HIPAA
- International:
 - UL-2900
 - ISO 27000 Series
 - NIST Cybersecurity Framework

7. Authors

This publication has been written and reviewed by:



Gilles Forconi
Software quality Manager
g.forconi@debiotech.com



Laurent Colloud
Software Project Manager
l.colloud@debiotech.com



João Budzinski
R&D Director
j.budzinski@debiotech.com

This document has been written for information purposes only. Debiotech or its authors cannot be held responsible in any way.

Debiotech is glad to share its knowledge with innovative companies from the MedTech industry. Your feedbacks on this publication are welcome and will be used to update it or to create new publications on topics you care about.

8. Next steps

contact us directly for personalized support : contact@debiotech.com

Learn more thanks Debiotech's website
<https://www.debiotech.com>



Follow Debiotech's activities on LinkedIn
<https://www.linkedin.com/company/debiotech-sa>





DEBIOTECH



**YOUR STRATEGIC PARTNER
FOR MEDICAL INNOVATION**

30+ years of expertise at your service.

Software: Digital Health, Embedded, SaMD, Cybersecurity, AI

Electronics: Design, Verification and Validation

Mechanics: Design for micro-fabrication & fluidics systems

Supply chain development and optimization