X-by-Construction Design Framework for Engineering Autonomous and Distributed Real-time Embedded Software Systems

Research and Innovation Actions

Horizon 2020, Topic ICT-50-2020:
Software Technologies

Grant agreement ID: 957210

– Whitepaper –

XANDAR Security Processes

31 December 2023
Version 1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957210.

Document information

Document title:	Whitepaper – XANDAR Security Processes
Author(s):	Henry Hui Sena Yengec Tasdemir Fahad Siddiqui Sakir Sezer Kieran McLaughlin Kammy Fong Balmukund Sonigara
Version:	1.0
Date:	November 2023
Dissemination level:	Public

XANDAR consortium

No.	Short name	Name	Country
1	KIT	Karlsruher Institut für Technologie	Germany
2	UOP	University of Peloponnese	Greece
3	DLR	Deutsches Zentrum für Luft- und Raumfahrt	Germany
4	AVN	AVN Innovative Technology Solutions Limited	Cyprus
5	VECTOR	Vector Informatik GmbH	Germany
6	BMW	Bayerische Motoren Werke Aktiengesellschaft	Germany
7	QUB	The Queen's University of Belfast	United Kingdom
8	FENTISS	Fent Innovative Software Solutions SL	Spain

Executive summary

The EU X2020 XANDAR project undertakes research in the development of a software toolchain, based on the X-by-Construction (XbC) paradigm, to streamline the complex, error-prone development process of safety-critical embedded system. Furthermore, the XANDAR XbC toolchain provides extensive runtime safety and security features such as system health monitoring, illegal access, intrusion, and privilege violation.

This whitepaper provides an introduction on the security process provided by the toolchain. Starting from the modelling phase, a Cybersecurity Engineering process provides a safety-aware approach on creating secured software components. The paper also demonstrates the feasibility on automating the development of security software components with security pattens. After a safety-critical embedded system is developed, it is managed at runtime by the security lifecycle management process, where the onboarding, update, and decommission of software components is closely monitored. The approach for runtime monitoring is facilitated by a runtime security monitoring architecture.

The XANDAR security process is a holistic approach where the threats towards safety-critical embedded systems are carefully considered. Security is integrated during system design and spans the entire operation lifecycle, where the secured decommissioning of the components is also addressed.

Table of contents

1 Introduction

"X-by-Construction Design framework for Engineering Autonomous & Distributed Real-time Embedded Software Systems", or XANDAR is an EU Horizon 2020 project dedicated to providing a software toolset to address the changing landscape of safety-critical embedded systems. In the XANDAR project, two use cases of interest are studied: avionic and automotive.

A major goal of the project is to address the need of greater system resiliency due to the increasing adaptation of networked embedded systems. This also means establishing and maintaining system trustworthiness while assuring system safety and security. The XANDAR toolchain ensures the system and safety requirements established during the initial modelling process are upheld. This is done by using automatic code generation, model-in-the-loop simulation, verification, and validation functionalities. Moreover, the use of hypervisor technologies in the XANDAR runtime system ensures that system resources, e.g. processing time and memories, can be allocated to individual safety-critical components where segregation can be guaranteed.

In terms of security, naturally, threats associated with the increased connectivity must be addressed. In the XANDAR project, we consider this issue not only being a task for the developer of the embedded software, but also an effort throughout the entire lifecycle of the system. The XANDAR toolchain assists the development process of embedded software by proposing a cybersecurity engineering process and the introduction of security patterns. On the other hand, to protect the runtime of safety-critical systems, a security lifecycle framework is introduced. There are four unique steps in the security lifecycle: secured onboarding, security monitoring, security software update, and secured offboarding. Each of this security components addresses some specific issues on a safety-critical system.

Conceptually, imagine each safety critical system being a castle that protect personnel and important properties within its parameters. The fortification of the castle is created methodologically (cybersecurity engineering process) using sturdy materials (security patterns). The introduction of new personnel or any changes to it has to be properly onboarded and updated (security onboarding and security software update). When the castle is under its daily operations, all activities are being monitored constantly by guards and lookout towers (Runtime monitoring). Finally, personnel exiting the castle has to be checked in case information or properties are being exfiltrated that may inhabit the normal functions of the castle (Secured offboarding). The XANDAR project made suggestions for the extensive research topic described, with focuses specifically on security patterns and security lifecycle management, more details will be provided in this whitepaper.

2 Threats against Safety-Critical Embedded Systems

To effectively defend safety-critical embedded systems from intruders, it is essential to understand the threats against them. As part of the XANDAR research, a wide spectrum of security threats was identified. Figure 2-1 summarised these threats into five vectors: human, conflict with safety requirements, supply chain, system, and future technologies [1]–[6]. It is worth highlighting that, among the top ten emerging cybersecurity threats and challenges for 2030 published by The European Union Agency for Cybersecurity (ENISA) [1], eight of the threats can be directly applied on safety critical systems, as marked by ⁺ in Figure 2-1.

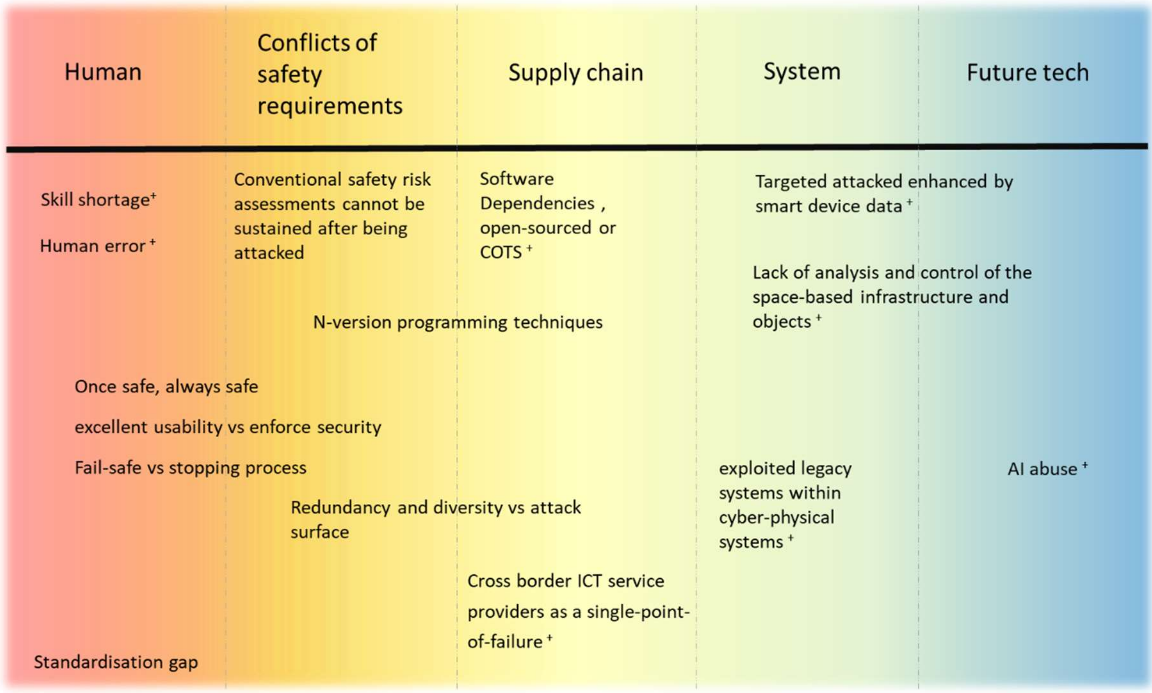

Figure 2-1: Spectrum of threats against safety critical systems ((⁺): ENISA top 10 emerging cybersecurity threats for 2030 [1])

Safety-critical systems, with the current technologies, are developed by humans. The increased complexity of these systems necessitated a higher degree of specialisation from developers and engineers. However, even with the expertise, it is still an error-prone process because of the complexity. Professional or legal standards often improve the safety and security of systems, however, it has been pointed out by ENISA that gaps remain in the current standardisation [4].

One of the significant factors for the rising complexity is that traditional safety engineering requirements conflicts with security best practise. Specifically, safety, once properly enforced, can be guaranteed if the systems or the physical environment remain unchanged, which is not the case for security. Safety systems require prompt response with minimum interactions and fail-safe mechanisms to guarantee the availability of the process. Some sophisticated safety systems also require redundancy by hardware diversity and N-version programming techniques (multiple versions of the same software that are developed differently). In contrast, the ever-changing security landscape require constant changes to the system. Additionally, the functionality (e.g., safety) of a system can no longer be trusted after an attack. Also, enhanced security measures usually introduce extra response time or requires human interactions, e.g., encryption and multi-factor authentication. Moreover, a general goal for a security mechanism is to terminate a compromised process. It is also best practice to reduce the variation of different hardware or software components in a system to reduce the attack surface. Software or hardware dependencies, whether it is open-sourced or commercial-off-the-shelves, will unavoidably contain security threats. This problem can be worsened if the solution providers are out of the system owners' place of origin.

Another factor that greatly increase the complexity of safety-critical systems is the requirement of external input, like sensors readings or external communication. In the system point of view, it is difficult to guarantee the integrity of these external inputs, especially in the space domain. Similar to the space domain, Industrial Control Systems (ICS) are also under threats since these systems were built with now legacy components, where these components have little to no security built in. On the other hand, more advanced threats, such as the misuse of Artificial Intelligence, introduced great uncertainties to safety-critical systems.

To address the entire spectrum of threats towards safety-critical systems, it requires continuous efforts from the entire research community, the industries, and the governments. However, a shortcut for the solution perhaps is to provide tools for developers and engineers to create reliable, safe, and secured software.

2.1 Problem Statements

The goal of the XANDAR project is to improve the abilities of developers and engineers on creating safer and more secure systems. Therefore, the aim of this work is to addresses the following problems.

(1) System developers are required to incorporate security codes (e.g., encryption, digital signatures) in the embedded software. However, this is a non-trivial task since it requires certain level of know-how from the developers.

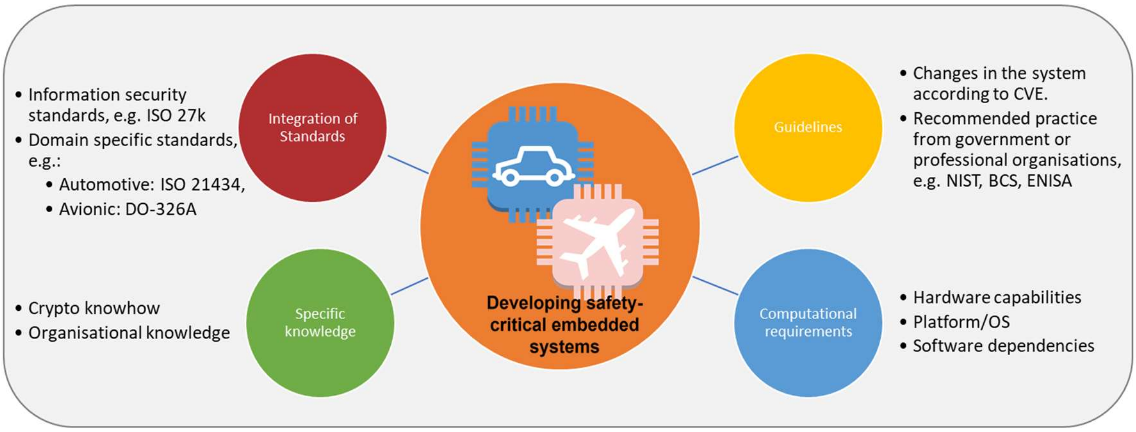

Figure 2-2: Example of the requirements engineers need to be aware of when creating safety-critical systems

In the ENISA's investigations on IoT standards gaps [4], it is stated that a complete approach towards IoT security can be found in a series of standards, but further work is needed to bring all standards together. Similarly, when developing safety-critical systems, although the complex security requirements can potentially be overcome with existing knowledge, developers have to be equipped with the knowledge of a wide range of topics. In addition, there are often computational limitations imposed when a system is implemented on a different platform. These are illustrated in Figure 2-2.

(2) *Even if an embedded system is designed to be secured, threats still exist during system runtime. This is mainly due to the ever-evolving scene of software or hardware vulnerabilities, in particular zero-days.*

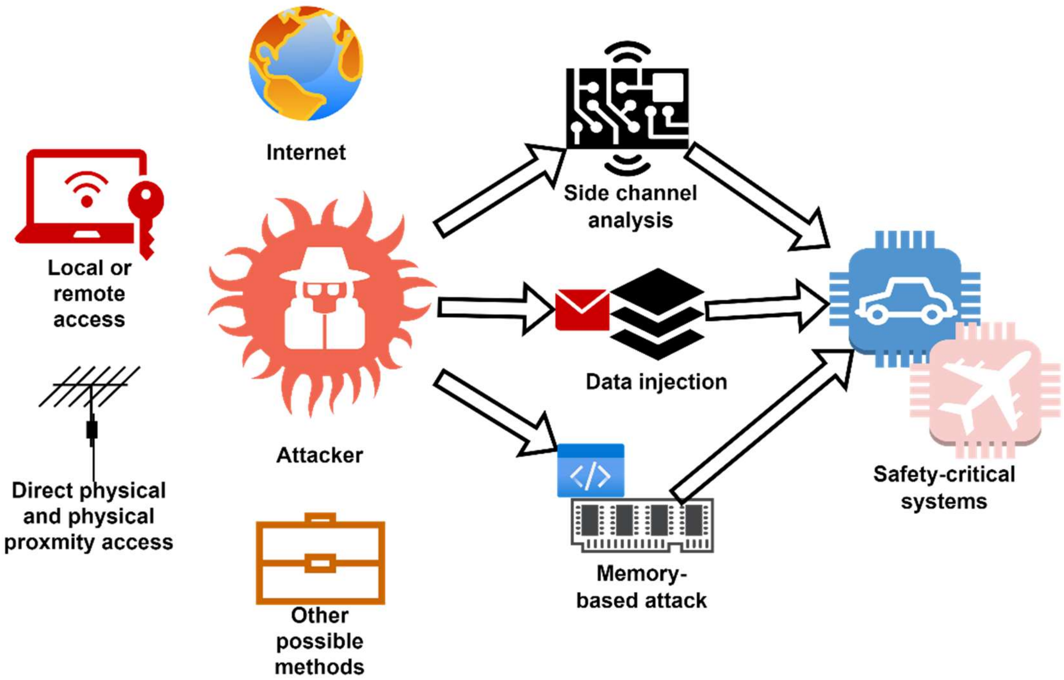

Figure 2-3: Runtime threats on safety-critical systems

Some of the threats targeting embedded systems are categorised by [7]. Figure 2-3 illustrates the attack vectors on safety critical systems. An attacker with access to a system, e.g., from the internet, can perform malicious actions. The actions that are relevant to safety-critical systems can be summarised into three categories: side-channel attacks [8], data injection attacks [7], [9] and memory-based code injection [10]–[13].

3 XANDAR Security Processes

3.1 Cybersecurity Engineering Process

As part of the XANDAR project, a six-step cybersecurity engineering process is proposed and the detailed description can be found in [14]. In essence, upon the availability of the systems requirements, the non-functional requirements (e.g., safety and security) are scoped and refined into specific objectives (① IDENTIFY). This facilitates the threat modelling and risk assessment (② ASSESS) that creates the specifications for safety and security controls. [15], [16] were published in collaboration with XANDAR partners to bridge the gaps between the Threat Analysis and Risk Assessment (TARA) with security standards of the XANDAR use cases, DO-326A/ED-202A and ISO/SAE 21434 for avionic and automotive respectively. The process creates threat models for information flow and identify vulnerabilities of components using existing tools, such as Microsoft threat modelling tool [17] and ThreatModeler [18]. Following this, the TARA framework is applied where the mitigation strategies are identified. This output of the risk assessment process can then be provided to step ③.

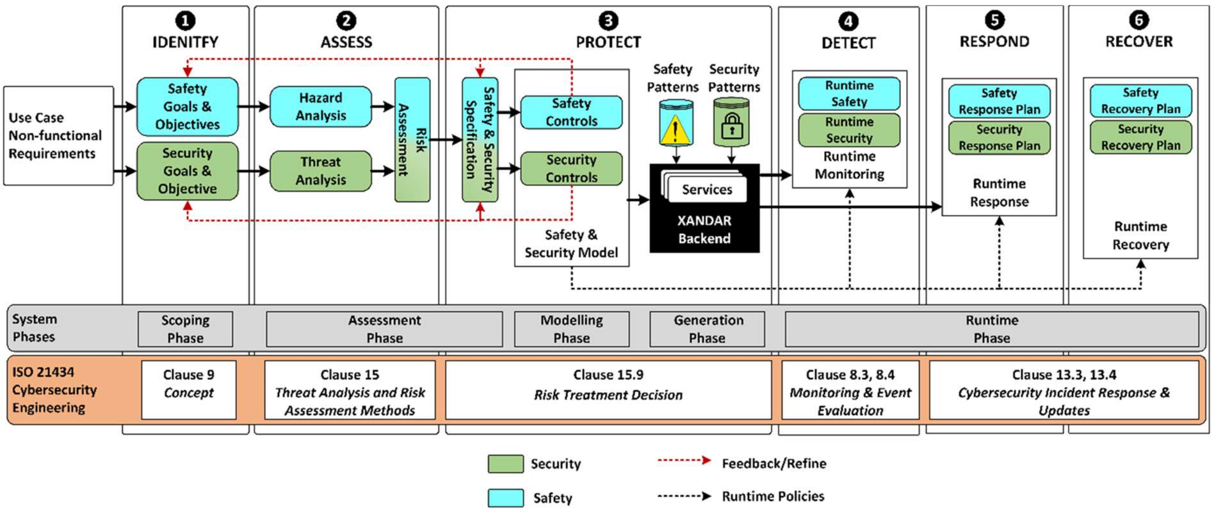

Figure 3-1: XANDAR security engineering process for safety-critical systems

The first part of the step 3 (③ PROTECT), is to create the specification for the safety/security controls. The previous output from the TARA process provides a great indication on the creation of the safety-aware security specification.

The second part of step 3 is the application of the safety and security patterns to aid the development of the safety and security components of the embedded software. Patterns are one of the foundations of the XANDAR toolchain, which replaces the manual implementation of functionality with automatic tasks. Most of the security tasks can potentially be automated: encryptions/decryptions, signatures and certificates generation and verifications, security tokens, pseudo-random number generator etc. However, significant research is required for the automated pattern to be created and applied correctly. In the XANDAR project, as a proof-of-concept, three security patterns are demonstrated: symmetric key encryption, digital signature, and hashed message authentication code, these will be documented in section 4. Security patterns address the problems in the manual traditional development process where developers have to implement all the security components, which is an error-prone process and require the understanding of the complicated security best-practises and the often-mandatory security requirements.

The remaining stages in the engineering process require the establishment of the runtime monitoring systems (④ DETECT), incidence response planning (⑤ Response) and recovery process to ensure the safety-critical system components can function normally (⑥ RECOVER). In XANDAR, this relies on the proposed security lifecycle management process.

3.2 Security Lifecycle Management

In XANDAR, the proposed security lifecycle management has four steps: secure onboarding, security monitoring, secure update, and secure offboarding, as depicted in Figure 3-2. This process assumes the availability of a hardware Trusted Platform Module (TPM) and the secure boot process. TPMs provide the cryptographic facilities required to ensure the integrity of the binaries of the kernel and the keys that can be used to establish trust on software. Due to the importance, they have become commodities for embedded systems in the recent decade.

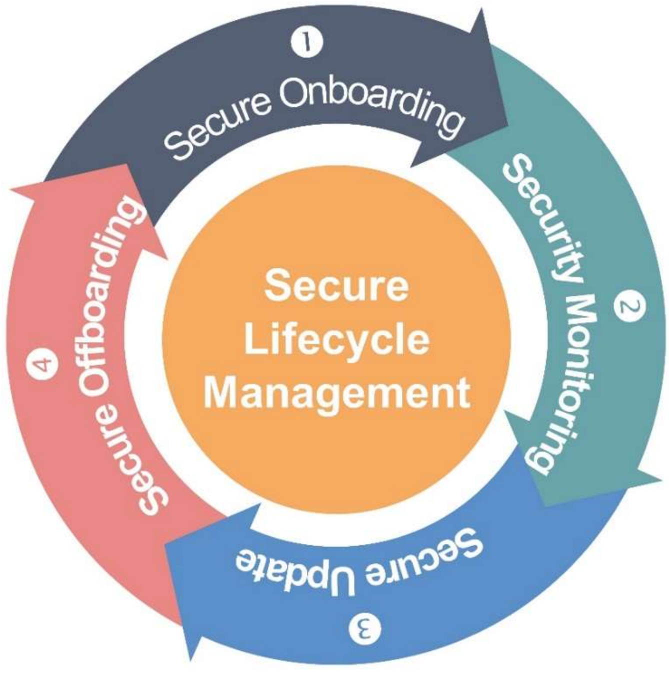

Figure 3-2: XANDAR security lifecycle management

3.2.1 Secure Onboarding

The introduction of new software or data into the system is a critical juncture that demands increased security. Central to this is the concept of secure onboarding, a method specifically devised to securely incorporate software components into systems, especially when these components may have external origins, such as from the internet or during initial software setups.

Secure onboarding, grounded in well-established security patterns, emphasizes three main components:

- Encryption/Decryption Mechanisms: One fundamental aspect is confidentiality. To achieve this, encryption mechanisms are employed to protect the data, ensuring it remains inaccessible to unauthorized users. Prior to deployment, decryption processes are applied, facilitating data retrieval in its original form.
- Hash Value Calculation: Data integrity, defined as the consistency and accuracy of data over its lifecycle, is verified by calculating the software image's hash values. This acts as a validation measure, confirming the data's untouched state.
- Authentication: Ensuring that data or software is genuine and originates from a trusted source is vital. This is realized through robust authentication processes, which determine the legitimacy of the onboarded data or software.

The onboarding procedure commences with an examination of the privileges associated with the partitions initiating the process, acting as the first line of defence against unauthorized installations, e.g. from malware. Subsequent to this foundational step, the emphasis is on ensuring the authenticity and integrity of the software. Authenticity is determined through signature verification. If the signature of the partition is validated, the system proceeds to calculate and compare hash values of the software, further reinforcing its integrity. Upon successful validation of both signature and hash values, decryption of the software is performed. Concluding the process, the system assesses and assigns the appropriate access privileges to the successfully onboarded software.

3.2.2 Runtime Monitoring

Runtime security monitors provide protection to safety-critical systems with constant detection of abnormal system behaviour and the ability to deal with security intrusion. The spatial and temporal separation provided by hypervisor technologies mitigates some of the threats. However, as mentioned in section 2.1, zero-day attacks remain a considerable risk even for the widely used or properly maintained hypervisors. Therefore, Chapter 5 is dedicated the XANDAR runtime security architecture and how it can be used to protect safety critical systems.

3.2.3 Feature on Demand Over-the-Air Updates

Feature on Demand Over-the-Air Updates offers users the flexibility to personalize their device experiences by enabling specific features, FoD leverages OTA updates. This dynamic provisioning, however, necessitates stringent security. As such, the encryption and hash value calculation mechanisms mentioned earlier are indispensable in ensuring the confidentiality, integrity, and authenticity of these updates.

The UNECE R156 regulation provides an outline on how vehicles software updates should be implemented. Moreover, ISO 24089 is currently under development to provide standards for software updates [44]. The process of OTA updates is composed of three stages, which are: Creation, Distribution, and Installation. Upon the creation of the update, it is encrypted and signed before it is distributed, e.g. through a secured channel on the internet. The installation process is similar to the secure onboarding process, where signatures are verified before the decryption of the payload. However, in addition, the integrity of the originally installed software should be verified prior to the update.

3.2.4 Offboarding

Offboarding process is the decommissioning of the installed software in the system. The major security risks are the unauthorised removal of a software component. Therefore, access privileges must be confirmed during this process. Moreover, critical functionalities should never be removed while the system is under normal working condition, even if it is intended as part of an upgrade process (e.g. removal of an old collision avoidance software in favour of an enhanced version with new hardware dependencies).

4 Security Patterns

Security patterns are fundamental for incorporating non-function security features into the XANDAR XbC embedded design paradigm. As reusable solutions to recurrent security challenges, security patterns facilitate a consistent and reliable method to address potential vulnerabilities.

Within the context of XANDAR, security patterns facilitate the process of selection, generation, and deployment of system-level security controls across networked embedded systems. These security patterns will serve as a bridge between system security modelling and runtime phase of system design and development lifecycle providing:

- The flexible realisation of XANDAR holistic risk-oriented safety and security processes that are defined based on security best practices, guidelines, and standards.
- Essential protection, detection, response, and recovery functions to the networked embedded system back to a secure state in case of software malfunction or launch of a malicious attack.
- To help software developer of automotive and avionics systems to leverage domain-specific security and safety expertise by transparently integrating security patterns with software runnable.

The pattern-based security approach enables software developers to incorporate security into their designs using automatic code generation tools, even without specialized security expertise. These system design processes are often manual, time-consuming, and error-prone, involving multiple stakeholders. The heterogeneity of safety and security controls, coupled with the lack of formalism, further complicates their selection, configuration, and deployment, requiring domain-specific expertise.

Automation is ideal to reduce errors, minimise human involvement, alleviate the workload, and enhance system resilience. Formal representation of these security and safety controls as patterns facilitates this approach, allowing developers to incorporate safety and security using automatic code generation tools.

To enhance the security of a safety-critical software function, the application developer makes use of security services (secure on-boarding, secure off-boarding, remote attestation etc.) generally provided by the platform manufacturer. These security services, under the hood, use various cryptographic algorithms (AES256, SHA256. RSA-2048 etc.) that serve as building blocks, to deliver necessary data security functions. Nevertheless, adding security to these safety-critical software functions either manually or through automatic code generation method comes at a cost. One of these costs is the timing overhead, which depends on the computational complexity of the used cryptographic algorithm, the control messaging (key exchange, certificate generation etc.) and the data size. A separate work specifically investigating this can be found on IEEE Xplore [19].

4.1 Data Confidentiality

Table 4-1 describes a data confidentiality pattern. Within the safety-critical systems, a data message is required to pass through one or more intermediaries. While in transit, there is a risk that an attacker can eavesdrop on these data messages between the sender and the receiver using malicious software or exploiting known/unknown system hardware/software vulnerabilities. Thus, compromising the confidentiality of data messages. This disclosed information further can be used to launch a range of malicious attacks such as malicious manipulation of data to adversely alter the behaviour of the system.

Table 4-1: Data confidentiality security pattern

Characteristic	Description
Pattern Name	Data Confidentiality
Pattern Type	Hardware and Software
Problem	How to prevent disclosure of critical system data-at-rest and data-in-motion from not intended recipients?
Solution	The data and message contents shall be encrypted ensuring that only intended recipients can access the critical system control and data.

4.2 Data Authenticity

Table 4-2 describes a data authenticity pattern. Within the safety-critical system, a data message is required to pass through a communication channel. While in transit, there is a risk that an attacker can adversely modify the content of a data message originated from a legitimate source by exploiting known/unknown system hardware/software vulnerabilities and malicious software. Thus, compromising both integrity and authenticity of a data message. This unauthorized data message modification can be used to launch a wide range of malicious attacks such as loading and execution of malicious code, unauthorized access to resources, denial-of-service, elevation of privileges etc. to gain control of the system and adversely alter the behaviour of the system.

Table 4-2: Data authenticity security pattern

Characteristic	Description
Pattern Name	Data Authenticity
Pattern Type	Hardware and Software
Problem	How to ensure authenticity and protect integrity of critical system data-at-rest and data-in-motion against unauthorized modification?
Solution	The data and message contents shall be cryptographically hashed and signed by the sender to detect unauthorized modification of critical system data message and verify the source of data message by the recipients.

4.3 Data Integrity

Table 4-3 describes a data Integrity pattern. Within the safety-critical systems, a data message is required to pass through a communication channel. While in transit, there is a risk that an attacker can adversely manipulate/tamper the data message between the sender and the receiver using malicious software or exploiting known/unknown system hardware/software vulnerabilities. Thus, compromising the integrity of a data message. This tampered data message can be used to launch a range of malicious attacks such as man-in-the-middle, spoofing, denial-of-service, replay attacks etc. to adversely alter the behaviour of the system.

Table 4-3: Data Integrity security pattern

Characteristic	Description
Pattern Name	Data Integrity
Pattern Type	Hardware and Software
Problem	How to protect integrity of critical system data-at-rest or data-in-motion against manipulation and tampering?
Solution	The data and message contents shall be cryptographically hashed to detect adverse manipulation/tampering of critical system control and data by the recipients.

4.4 Security Patterns & Algorithms Within the Scope of XANDAR

Table 4-4 provides an example on which cryptographic algorithms are available and are ready to be utilised. Security patterns only allow standardised cryptographic algorithms that are well tested and recommended by international standardisation organisations including ISO, IEC, ETSI, NIST, etc. Therefore, the main differences between selection depend on the computational resources available to a given platform. As mentioned previously, a benchmark of the different algorithms is provided in [19].

Table 4-4: Associated cryptographic algorithm to each security pattern.

Security Pattern	Cryptographic Algorithm
Data Confidentiality	AES128, AES256, Camellia with CBC, CFB, OFB, CTR, and ECB modes of operation.
Data Integrity	SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512
Data Authenticity	RSA-2048, Ed-25519, Ed-448, ECC-DSA/SECP256R1, SECP384R1, SECP521R1, SECP224R1, SECP192R1, SECP256K1, HMAC-SHA256, HMAC-SHA384, HMAC-SHA512

5 Runtime security monitoring

Having a well-fortified castle can only protect the critical asset from external threats. When the adversaries managed to create a tunnel into the fortress, without internal protection, the castle can still be compromised. In safety-critical systems, this tunnel can be the zero-day exploits of the system. This is when security monitoring (security guards) and security components (a watchtower inside the caste) are necessary. This section provides an overview of the Runtime Security Monitoring (RSM) architecture proposed in the XANDAR project.

In the XANDAR toolchain, hypervisor technologies are utilised for the system runtime to provide operational separation between safety-critical software, where the toolchain design process guarantees the timely execution of the critical software component. In terms of security, the spatial separation provided by hypervisors, in theory, segregate safety-critical software components from any malicious components on the system. However, there is a need to further protect these systems against hardware-based vulnerabilities and/or zero days, such as Spectre [20] and Meltdown [21]. These vulnerabilities allow adversaries to by-pass any privilege checks on memory access and execute code.

In the safety-critical context, Airbus [22] has published a security model for distributed critical systems that segregates a network of critical facilities into different trust levels and how the network can be monitored. This section presents a security architecture that can be implemented on an embedded system, with the help of hypervisors.

A few assumptions are made for the proposed mechanism:

I. A hardware trusted platform module and secure boot functionality is available.
II. Availability of a trusted service repository for critical and safety related software (typically from vendors).
III. Firmware updates will not be applied during normal operations.

Figure 5-1 shows the overall proposed architecture. Different software components are assigned to different enclaves. There are two types of hypervisor enclaves: System Enclaves and User Enclaves. Each enclave contains partitions in the form of bare-metal applications or virtual machines (VM). The SE houses critical components of the safety-critical systems while the UE are intended for the optional components. The proposed architecture must include two SE by default, the Security Enclave (SE) and the Critical systems Enclave (CE). For optional and user specific components, there are two-level of optional UE available: the UE-I and UE-II. In addition, a specific security concept is introduced in the XANDAR RSM: Data-at-rest (DAR) security. DAR refers to data that is either stored in the volatile memory or persistent storage. This can be applications, telemetry data, user data, or cryptographic keys. This information should be encrypted in the system. Detailed information of the architecture is provided in the following sections.

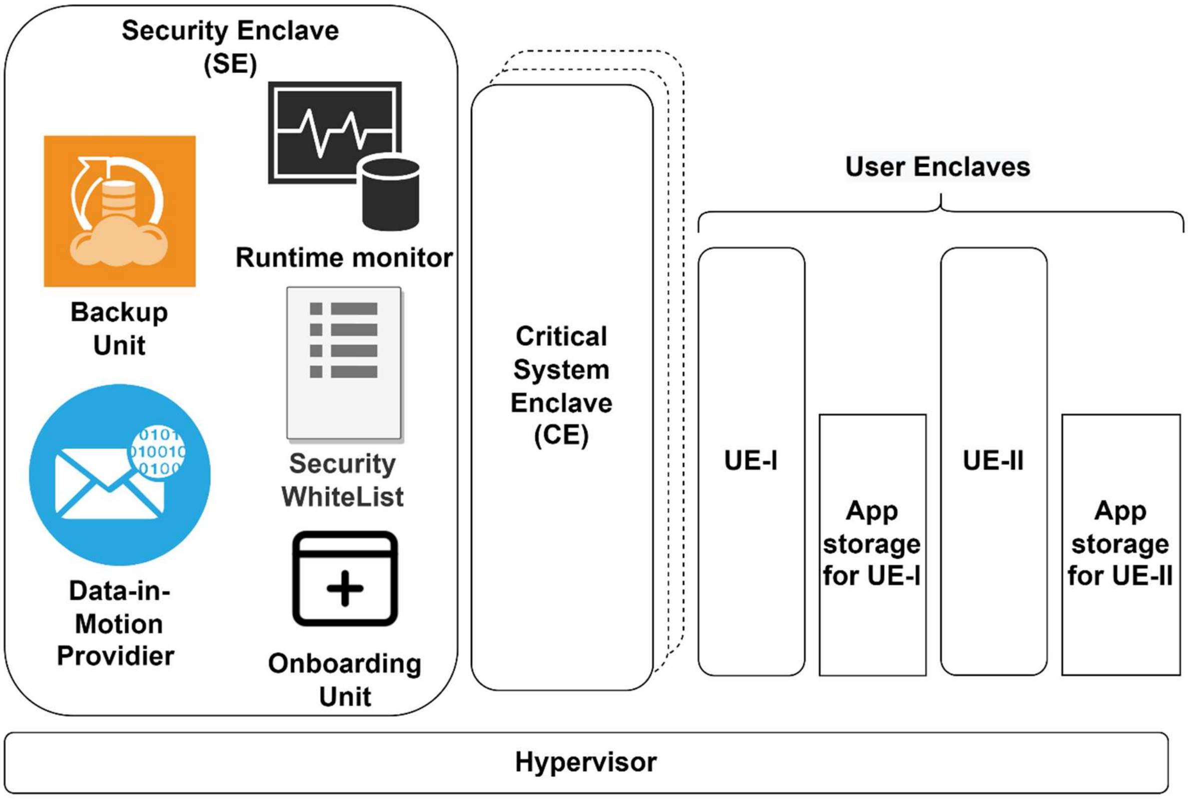

Figure 5-1: XANDAR runtime security monitoring architecture

5.1 User Enclaves

UE-I and UE-II are the sub-type of UEs that serve different purposes for users. UE-I provides functionality that interacts with the hardware of the systems, for example, custom functionality or features-on-demand (e.g., enabling driver assistance system). UE-I is where users can choose non-critical system functions provided by vendors or trusted third parties. UE-II on the other hand provides limited access to the hardware and the privilege can be revoked anytime by the runtime monitor, e.g., the infotainment system of a self-driving vehicle where the screen access can be revoked on safety or security events in favour of important actions. In addition, if a partition in any UE is a VM, an associated Application Storage (AS) is required. This allows the separation of the VM image and installed application, where the invocation of an application requires explicit capabilities. This will be enforced by the runtime monitor, which has the ability to stop potential execution of malicious applications and remove or restore access to the AS at will.

5.2 Critical System Enclave (CE)

The CE encapsulates the essential component of a safety-critical embedded system, for example, the brake system of an autonomous road vehicle or the collision avoidance system of an unmanned avionic vehicle. A major restriction on the CE is the inability to perform changes to the included binaries during runtime. The justification of this decision is perhaps obvious; however, this also calls for additional validation and verification before the system is deployed to ensure functional correctness. The integrity of the CE is also examined regularly. Other behaviours of the CE are also strictly monitored by the Security Enclaves, for example, during maintenance, the installation or the updating of new software be performed by the onboarding partition in the SE, and the communications with all components are monitored by the Data-In-Motion provider.

5.3 Security Enclave (SE)

The SE contains a collection of security components provided by the proposed architecture.

5.3.1 Data-In-Motion (DIM) Provider

Data-In-Motion refers to the transmission of data from one endpoint to another, either internal (between software services) or external (to the internet). The DIM provider provides the cryptographic components that facilitate secure inter-component and external communications. Both data encryption and data signatures may be enforced in each communication when required. In the XANDAR context, the focus remains in the Inter-Process Communication (IPC), since secured external communication often relies on public-key infrastructure and it is a well-researched topic with established solutions.

The DIM provider manages the IPCs and ensures access from authorised users. Moreover, it has the ability to protect the system against reconnaissance and spoofing from an infected component by potentially directing all IPC communication to the DIM provider. This is especially beneficial for communication between critical and non-critical software components. Since the IPC redirection implies that, unless it is intended by design, the sender of the IPC has no knowledge of other components and functionalities that exist in the system. Moreover, when the IPCs are encrypted and signed, even if a malicious component gained access to the system's memory, the details of the IPC cannot be obtained or amended.

5.3.2 Onboarding Unit (OU)

The OU is used to facilitate the onboarding, over-the-air updating and offboarding process in the security lifecycle management process. This can be user-installed software or system firmware updates. As mentioned earlier, it is assumed that firmware updates will only be applied outside of normal operations, e.g., during maintenance. On the other hand, the installation, removal, and updating of user software is handled by the OU. User software can be provided by the vendors or third parties through a trusted service repository. The DIM provider is responsible for the security of the communication, while the OU logs the details of the software being installed, e.g., the hash of the binary, and checks the system component for onboarding privileges. The same is also done by the OU when offboarding the unit to prevent unauthorised removal of software.

5.3.3 Security Whitelist

The security whitelist facilitates the initialisation of the system and security monitoring at runtime. Each entry of the whitelist includes the digital signature of the application binary and any access privilege information. Having obtained the keys provided by the hardware TPM and the secure boot process, the runtime monitor can verify via the digital signature the integrity of the binary on important events, like during system start-up and software updates. The access privilege information records which system components are allowed access to a particular binary.

5.3.4 Backup Unit (BU)

Safety-critical systems often require redundancy or a safety net. For example, [32] proposed a hardware redundancy measures for Unmanned Aerial Vehicles. The BU in this work provides a way to restore a system to the last known good state. Only the Runtime Monitor (RM) has the authority to invoke the BU as it is intended to be a last resort in the case that RM loses control of a system unit (e.g., a malware infected the UE-II and gained control of the CE and cannot be terminated). To achieve this, the BU manages a copy of the CE and the base images of UEs. In terms of the corresponding app storage, limited (due to resource constraint) important functionality in UE-I can be restored as part of a backup, while less important applications can either be re-onboarded according to the whitelist and the last known good state of the system, or simply be ignored. Given interruptions are likely to be caused by this action, system designers should thoroughly analyse and evaluate the priority of the safety and functional components, which should already be part of the normal design process. In the case where redundant hardware is available, e.g., as mentioned in [32], the system can be up and running again with minimal effect to its operation.

5.3.5 Runtime Monitor (RM)

The security architecture proposed shares the burdens and responsibilities to different components in the SE. Therefore, when it comes to runtime monitoring, a great portion of workload for the RM is the data acquisition from the components in the SE. For instance, the RM actively monitors the communication facilitated by the DIM provider, checks the OU for anomalies, and maintains the integrity of the security whitelist. In addition, the runtime monitor performs regular data-at-rest checks, monitors user enclaves for app invocation, and has the ability to terminate processes that are flagged abnormal in the UEs.

5.4 Summary

A novel architecture is presented which allows runtime monitoring to be directly deployed on safety-critical embedded systems. The architecture is designed to be generic and can be applied on different hypervisors. A major security benefit of the architecture is the ability to ensure the integrity of communication among components as well as tamper-proof data storage. In addition, by adopting a multi-enclave design as proposed, components with different criticality can be protected by access segregation.

6 Conclusion

The field of safety-critical embedded systems is undergoing rapid changes into automation. Security threats introduced risks to the embedded systems by undermining the safety components which impose threats to critical services and human lives. This whitepaper introduced the XANDAR's holistic approach to embedded security, where security is considered when designing, implementing, monitoring, and decommissioning safety-critical systems.

Security with XANDAR starts with a six-step cybersecurity engineering process to ensure the created safety-critical systems are safe and secure. This is assisted by the introduction of security patterns, which have the goal of automating the development of security components and reducing errors in the complex development process.

The security lifecycle management framework ensures the software components in safety-critical systems are correctly onboarded, updated, and offboarded. Within the security lifecycle, safety-critical systems are monitored at runtime to prevent zero-day threats. This is facilitated by the security components of the XANDAR runtime security monitoring architecture.

XANDAR's holistic approach to embedded security effectively amalgamates automated secure system design considerations at design-time and advanced system monitoring techniques at runtime. Experimental results based on use-case studies clearly demonstrate that the Xandar project is paving the path for next generation security architectures for future embedded systems.

7 References

[1] R. Mattioli, A. Malatras, E. N. Hunter, M. G. B. Penso, D. Bertram, and I. Neubert, 'Identifying Emerging Cyber Security Threats and Challenges for 2030', European Union Agency for Cybersecurity, Mar. 2023.

[2] R. kamal Kaur, B. Pandey, and L. K. Singh, 'Dependability analysis of safety critical systems: Issues and challenges', *Ann. Nucl. Energy*, vol. 120, pp. 127–154, 2018

[3] I. Lella, E. Tsekmezoglou, R. S. Naydenov, C. Ciobanu, A. Malatras, and M. Theocharidou, 'ENISA Threat Landscape 2022', European Union Agency for Cybersecurity, Oct. 2022.

[4] E. Andrukiewicz, S. Cadzow, and S. Górniak, 'IoT Security Standards Gap Analysis', European Union Agency for Cybersecurity, Dec. 2018.

[5] C. W. Johnson, 'Why we cannot (yet) ensure the cybersecurity of safety-critical systems', 2016 Accessed: Oct. 19, 2023. [Online]. Available: https://eprints.gla.ac.uk/130822/

[6] FREQUENTIS AG, 'White paper: System security in a safety critical context'. Apr. 2019.

[7] D. Papp, Z. Ma, and L. Buttyan, 'Embedded systems security: Threats, vulnerabilities, and attack taxonomy', in *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, Jul. 2015, pp. 145–152.

[8] F. Menichelli, R. Menicocci, M. Olivieri, and A. Trifiletti, 'High-Level Side-Channel Attack Modeling and Simulation for Security-Critical Systems on Chips', *IEEE Trans. Dependable Secure Comput.*, vol. 5, no. 3, pp. 164–176, Jul. 2008

[9] M. Hasan and S. Mohan, 'Protecting Actuators in Safety-Critical IoT Systems from Control Spoofing Attacks', in *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things - IoT S&P'19*, London, United Kingdom: ACM Press, 2019, pp. 8–14. Accessed: Nov. 17, 2022. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3338507.3358615

[10] H. Shacham, 'The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)', in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 552–561.

[11] R. Hund, T. Holz, and F. C. Freiling, 'Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms', p. 16

[12] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, 'Non-Control-Data Attacks Are Realistic Threats.', in *USENIX security symposium*, 2005, p. 146.

[13] H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena, and Z. Liang, 'Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks', in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 969–986.

[14] F. Siddiqui *et al.*, 'XANDAR: A holistic Cybersecurity Engineering Process for Safety-critical and Cyber-physical Systems', in *2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring)*, IEEE, 2022, pp. 1–5. Accessed: Oct. 23, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9860859/

[15] F. Siddiqui *et al.*, 'Cybersecurity engineering: bridging the security gaps in advanced automotive systems and ISO/SAE 21434', in *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, IEEE, 2023, pp. 1–6. Accessed: Oct. 23, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10200490/

[16] F. Siddiqui *et al.*, 'Cybersecurity engineering: bridging the security gaps in avionic architectures and DO-326A/ED-202A', in *AIAA/IEEE Digital Avionics Systems Conference (DASC)*, Institute of Electrical and Electronics Engineers Inc., 2023. Accessed: Oct. 23, 2023. [Online]. Available: https://pure.qub.ac.uk/en/publications/cybersecurity-engineering-bridging-the-security-gaps-in-avionic-a

[17] R. McRee, 'Microsoft threat modeling tool 2014: Identify & mitigate', *ISSA J.*, vol. 39, p. 42, 2014

[18] 'ThreatModeler - Automated Threat Modeling Solution', *ThreatModeler*. https://threatmodeler.com/ (accessed Oct. 31, 2023).

[19] S. B. Yengec-Tasdemir, F. Siddiqui, S. Sezer, H. Hui, K. McLaughlin, and B. Sonigara, 'A Comparative Analysis of Security Patterns for Enhancing Security in Safety-Critical

Systems', in *2023 IEEE 36th International System-on-Chip Conference (SOCC)*, IEEE, 2023, pp. 1–6. Accessed: Nov. 02, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10256708/?casa_token=ybEDMBi8gi-oAAAAA:akCgzrIrCUTkSm0EQ0hZ_-VV8wzIJ8ipU7NP3q-f1SvM2mXFs4myK22NCzMS6YWgMdFtM50ayUY

[20] P. Kocher *et al.*, 'Spectre attacks: Exploiting speculative execution', *Commun. ACM*, vol. 63, no. 7, pp. 93–101, 2020

[21] M. Lipp *et al.*, 'Meltdown: reading kernel memory from user space', *Commun. ACM*, vol. 63, no. 6, pp. 46–56, May 2020

[22] Y. RÜTSCHLÉ, 'A security model for distributed critical systems'. AIRBUS, 2021.