Zero Trust Architecture Approach for Developing Mission Critical Embedded Systems

Michael Vai[1], David Whelihan[1], Eric Simpson[1], Donato Kava[1], Alice Lee[1], Huy Nguyen[1], Jeffrey Hughes[1], Gabriel Torres[1], Jeffery Lim[1], Ben Nahill[1], Roger Khazan[1], and Fred Schneider[2]
[1]MIT Lincoln Laboratory, [2]Cornell University
PoC: mvai@ll.mit.edu

***Abstract*—This paper describes a Zero Trust Architecture (ZTA) approach for the survivability development of mission critical embedded systems. Designers could use ZTA as a systems analysis tool to explore the design space. The ZTA concept of "never trust, always verify" is being leveraged in the design process to guide the selection of security and resilience features for the codesign of functionality, performance, and survivability. The design example of a small drone for survivability is described along with the explanation of the ZTA approach.**

Keywords— Zero trust architecture; Security; Resilience; Survivability; Mission Assurance; Embedded System; Systems Analysis; Security by Design; Survivability by Design

I. Mission Critical Embedded Systems

In the context of this paper, an embedded system would be a high performance computing system with dedicated functionality and well-defined behavior [1]. An embedded computing system example is the mission control computer for a drone. This is in contrast to an enterprise computer system designed for general computing. Furthermore, an embedded system is often optimized for its specific functionality in terms of smaller form factor, lower power consumption, and/or higher throughput. For survivability, a mission (or safety) critical embedded system should be defended against attacks with security (i.e., hardening) technologies, and equipped with resilience properties for mission assurance when an "eliminated" failure nevertheless occurs. At the same time, the designer needs to minimize the impact of enhanced survivability on size, weight, and power (SWaP), usability, cost, and development schedule. We have been developing cyber security and resilience design methodologies and associated technologies for mission critical platforms [2-5].

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

The current work has been motivated by the current government-wide effort to migrate to Zero Trust Architecture (ZTA) [6, 7]. The mandate is currently directed at enterprise level computing systems, however, we have recognized that ZTA offers a new persepective of designing survivability into mission critical embedded systems. For clarity and completeness, we present the ZTA approach by itself, but it is most productive when applied together with the systems analysis methodologies described in [2-5].

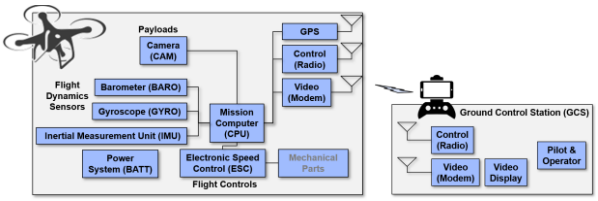
Historically, an embedded system (e.g., an airborne radar signal processor) is often implicitly trusted after its installation as it will remain unchanged for a long time, particularly when it is deployed in a protected location such as inside an airplane. This assumption no longer holds in today's embedded systems, which are increasingly more programmable, configurable, and upgradable in order to keep up with new technologies and defend against the latest threats. Future missions would require even more agile flexibility, sustainability, and upgradability.

Establishing effective survivability requirements for an embedded system is notoriously difficult. Providing evidence for meeting such requirements is harder yet, as it demands that we prove a negative. An embedded system that is designed to operate according to the zero trust tenet of "never trust, always verify" has numerous benefits for mission assurance. We have thus adapted a ZTA inspired survivability-by-design strategy in the development of mission critical embedded systems. Following this approach, the designer would incorporate technologies to establish a trust to proper system functionality and maintain that trust over its operation.

II. Embedded System Example: A Drone

Figure 1 provides an architectural overview of a small drone as a mission critical embedded system, which was previously discussed in [4] to demonstrate the designing of agility and resilience into embedded systems. We have reused the same design case in this paper to connect these design approaches. The drone architecture, CONOPS (Concept of Operations), its threats, and survivability features presented here are for illustration purposes only.

The drone shown in Figure 1a represents a typical embedded system, which can be viewed as a system of embedded systems, as each element in Figure 1a is a self-contained functional unit that includes its own microprocessor (or microcontroller) with dedicated software application for interfacing, control, and operation.



(a) (b)

Figure 1: Design example: (a) a drone with (b) a ground control station.

The drone contains a mission computer (CPU), which is the management control of the drone and is the focus of discussion in this paper. The payload of the system is a camera (CAM), which captures video streams. The drone includes a number of communications functions: a Global Positioning System (GPS) receiver for geo-location, a radio to communicate with the Ground Control Station (GCS) in Figure 1b, and a modem for streaming video. The CPU computes the flight dynamics using information from a few sensors: a barometer (BARO) for altitude measurement, a gyroscope (GYRO) for orientation and angular velocity measurement, and an inertial measurement unit (IMU) for acceleration measurement. The CPU, through an electronic speed control (ESC), controls mechanical parts such as motors, gears, etc.

III. SURVIABILITY-BY-DESIGN

Mission assurance requires all functions essential for mission execution to be available when they are needed. These mission essential functions must also be resilient to disruptions caused by either intentional (e.g., adversarial attacks) or unintentional (e.g., software bugs) causes. In the context of our discussion, security refers to incorporating technologies to harden a design against potential attacks. Resilience features, on the other hand, are provided in preparation for breakthroughs (i.e., when security has been compromised). The system needs to detect a disruption when it occurs, isolate it, and recover so that the mission could be continued. As we will explain below, it is always a good idea to bake in security and resilience features from the beginning so that their overheads on SWaP and performance could be accounted for.

A. *Real Time Survivability Requirement*

Figure 2 shows the dynamics of a mission critical operation and defines its real time survivability requirements. We have defined a few time parameters below to facilitate our discussion:

t_a: Time to cause failures by attack;
t_d: Time to detect and isolate attack;
t_r: Time to react and recover from the attack;
t_s: Mission dependent survivable time to react before unavoidable failure (e.g., a crash).

The goal of a mission critical embedded system design is to maximize attack time t_a and survivable time t_s and minimize recovery time ($t_d + t_r$). The best case scenario is that the system has been hardened properly to the attack and t_a is substantially longer than the expected mission time. No degradation would occur to the mission. However, the design may still need to deal with long term impacts by, for example, applying moving target technologies [9]. In contrast, the worst case happens when $t_d + t_r > t_s$. In this situation, the system has learned about the failure too late and taken too long to react. The system and thus the mission have failed.

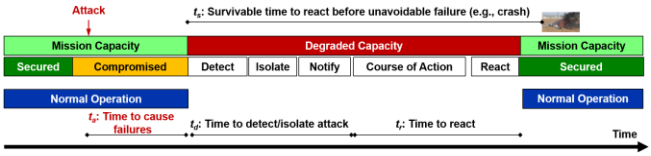


Figure 2: Mission assurance real-time operation.

Generally, the survivable condition is that $t_d + t_r$ is less than t_s. In this case while the mission will experience a disruption, but the system has adequate capability and resource to detect the failure, take a course of action in time, and return the system to normal operation.

B. *Zero Trust Architecture (ZTA)*

Zero trust is a set of principles that treats every component, service, and use of a system as continuously exposed to and potentially compromised by a malicious adversary [8]. At a high level, a ZTA depends on three attributes: compartmentalized access, continuous monitoring and adjustment, and applying security measures throughout the overall system. A ZTA enterprise system can thus support its intended mission by following the following zero trust security principles [8]:

- Identity verification – strong multi-factor user and device authentication;
- Access control – secure and approved access to resources;
- Resource protection – fine-grained control of approved resource utilization based on identity;
- Policy and orchestration – dynamic management of system use;
- Monitoring and analytics – analysis of system usage and security functions;
- Continuous operations – process to manage risks while supporting usability.

These principles are well associated with the need to enforce minimization, isolation, least privilege, monitoring, and recovery in the co-design of functionality, survivability, and performance.

Table 1: Association of ZTA principles with embedded system survivability design.

Zero Trust Principles	Embedded System Survivability Features	Example Survivability Technologies
Identity verification	Authenticate code, data, comms, command, etc.	Crypto
Access control	Secure and control access to data, resources, etc.	Separation kernel, Crypto
Resource protection	Manage HW/SW confidentiality, integrity, and availability	Crypto, secure boot, supply chain management
Policy and orchestration	Manage and protect system integrity	Policy enforcement of behaviors
Monitoring and analytics	Detect and isolate failures	Monitoring Service
Continuous operations	Recover with resilience, redundancy, etc.	Reboot into known good state

Table 1 associates the zero trust principles and survivability features available for embedded systems. For example, crypto is the technology for encryption, authentication, and verification. Separation kernel [e.g., 10] is for establishing software enclaves. Policy enforcement is for

the detection of abnormal behaviors. Monitoring service is to provide system observability.

IV. ZTA Embedded System Design Flow

We now explain the use of a zero trust survivability-by-design approach to codesign functionality and survivability so that security and resilience features are integrated as first class components into the system under design for mission assurance. The zero trust concept, which has been succinctly captured by the tenet of "never trust, always verify," would be used to assess the risks and vulnerabilities of all essential functions and components. The zero trust principles listed in Table 1 would then be used to guide the selection of security and resilience features.

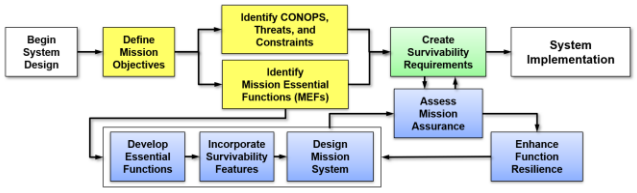


Figure 3: ZTA survivability-by-design process.

Figure 3 describes the process of this ZTA embedded system design flow. The explanation below has been presented with the drone in Figure 1 as an illustrative development example. The design activities and their products could be readily documented with a systems analysis and design diagram. Details of developing such diagrams were described in [4]. Figure 4 presents a subset of the analysis diagram generated for our drone development example, which we will explain as we describe the design steps.

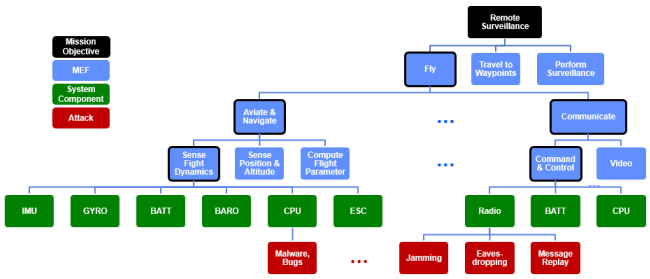


Figure 4: A systems analysis and design diagram showing the drone design example. MEFs (Mission Essential Functions) discussed here are highlighted.

A. Mission Objectives

The system design process begins with defining a mission objective. For our drone design example, the mission objective would be to perform remote video surveillance. This high level description will lead into the next step of identifying the concept of operations (CONOPS), threats, and constraints.

B. CONOPS, Threats, and Constraints

CONOPS refers to some details of the operational scenario and adds more specifics to the objective. One of the best ways to convey this is with usage vignettes. These are small stories that capture how the system will be used in a mission and what constitutes mission success. The vignettes should include who/what the mission elements are, how they interact, and what their responsibilities are during different mission phases.

In our development example, the GCS operator will send waypoints and targets to the drone over the radio. The drone will rely on autopilot to reach waypoints and perform video surveillance on targets. The modem will stream the target video to the GCS display.

The mission objective discussion with the stakeholder will produce a high level understanding of the system features or properties that are valuable to adversaries. These are useful because they convey at a high level what the mission stakeholders care about. The concerns usually relate to the loss or corruption of critical program information. The process will take such statements, explore potential threats, and derive how the threats could manifest themselves to impact the system.

A high-level concern of our design example is that an adversary may want to disrupt the video surveillance operation. Potential attacks include compromising the CPU through, for example, malware infection and interfering with communications between the GCS and the drone by, for example, radio jamming.

The constraints indicate how much design flexibility is possible to integrate security and resilience technologies. Constraints could be operational or technical, which are some of the most important information in the process. For example, an operational constraint could be the need to store classified information, which determines the minimum requirement of isolation and protection. Technical constraints could be the system's limited SWaP, or an inability to change legacy hardware or software. In our example, we assume that the drone has stringent SWaP and cost requirements. The flexibility of replacing commercial-off-the-shelf (COTS) components is also limited.

C. Mission Essential Functions

Mission essential functions (MEFs) are a set of high level functionality required to meet the CONOPS. We typically break down system mission objective into a few (3–5) high-level goals that can be succinctly described and codified as high-level MEFs. These will be decomposed hierarchically until the system hardware/software component boundary (i.e., when mission goals cease to be abstract) is reached. As shown in Figure 4, the mission objective of video surveillance will require MEFs of flying, traveling to waypoints, and performing surveillance. The MEFs are then developed with hardware and software components and interconnections for implementation. Figure 4 shows that flying has been decomposed into a couple of sub-levels until its dependence on system components becomes apparent.

The line between abstract, goal-oriented MEFs and system components (or subsystems) that provide functionality was fairly straightforward. What the system component row in Figure 4 misses, however, is the topology of the system. Topology, or the way subsystems are laid out and interconnected, is extremely important in a survivable system engineering effort because the connections between subsystems are usually also vectors for attack.

The outcome of this step is a full set of hardware and software elements, their interconnects, and components such as storage devices. These system components are shown in Figure 4 as a set of resources upon which the abstract MEFs depend. For example, in our example, sensing flight dynamics will depend on the CPU, IMU, GYRO, BATT, BARO, and ESC. An interconnect such as a bus (not shown in Figure 4) will also be needed. The full set of system components and how they are shared or accessible is a critical aspect of the design that is used in the next step to determine failure points and how they affect the overall system survivability.

Now we determine and capture failure or disruption states of the MEFs at the hardware/software level. In this step, the identified system components are cross-referenced with threats to identify how failures of those sub-systems can impact mission goals. Figure 4 indicates that the CPU is associated with potential malware attacks and the radio is associated with jamming, eaves-dropping, and message replay.

All system components are further annotated to include a list of cyber-induced failures under the categories of confidentiality, integrity, and availability. For example, if the CPU is rendered unavailable by malware infection, it is considered a single failure possibility of that resource. The fact that the CPU serves multiple MEFs means that its failure will be amplified. This is in contrast with a failure that only impacts a single MEF it was directly relevant to and will potentially have less impact on the overall survivability design. This is useful because it helps to direct the efforts of the design team to particularly impactful vulnerabilities.

D. Survivability Requirements

The survivability requirements need to clearly state what must be protected (e.g., functionality, critical program information, etc.). Ideally, all threat induced MEF failures and disruptions shall be prevented. In general, we would want to consider mitigating the impacts of failures instead of the attacks themselves. For example, radio jamming attack causes loss of radio service (availability impact) and malware causes the CPU to produce erroneous results (integrity impact). Instead of creating requirements to prevent/mitigate radio jamming and computing errors, the system may be further hardened into designing for radio availability and computing integrity. This practice tends to create more demanding requirements, but it could also lead to the design of stronger systems.

In our design example, an example survivability requirement could be that flight parameters shall be available at all time, which implies that the system must ensure, among other things, the integrity and availability of radio communications and the correctness of computing. The outcome of this process is a set of system survivability requirements that addresses the needs of the program stakeholders. It will be instrumental in the security technology selection activity and will drive the formulation of test criteria.

Based on the requirements, MEFs are created with security and resilience features incorporated and integrated into a mission system design. The first cyber defense should always be good cyber design practice, including open design, minimization of interfaces and code, complete mediation of access, and adherence to the Principle of Least Privilege [13].

Determining which technologies will decrease failures as the requirements demand, but are still within the box drawn by the system constraints, may require significant compromise. For example, constraints may dictate that there is not enough SWaP to encrypt the video downlink using approved encryption modes. A solution may be to use a lightweight cipher that will protect the data only up to its level of importance. The key to justifying such a decision is to provide evidence that the alternate cyber-technology is adequate and that there was no other way to meet the requirement. The backend of this process is designed to make the designer justify their choices and leave a record of them.

E. Mission System

The goal is first to reduce the number of cyber induced failures by buttressing security with techniques such as cryptography and separation, and second to be resilient to any failures that happen despite these efforts.

In our design example [2], the CPU was designed with security and resilience features, including secure boot, redundancy, separation kernel, monitoring service, policy enforcement, and recovery service. Cryptography has also been applied to protect communications over the radio [11]. Figure 5 shows the association of zero trust principles to the CPU design.

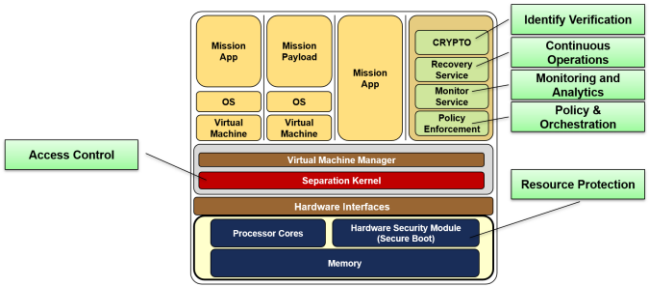


Figure 5: A mission computer designed with the called out zero trust principles.

F. Mission Assurance Assessment

The reduction of unmitigated failures is the key to increase mission assurance. However, proof of elimination of cyber induced failures is difficult or impossible to come by. Therefore, a second metric of assessing mission assurance is to label failures, regardless of whether they have been eliminated, as either detectable, isolatable, or recoverable. Recall from our discussion with respect to Figure 2, this consideration acknowledges that, despite our best efforts, a breakthrough can still be induced, and specifies whether it can be detected and recovered from in a timeframe that is acceptable with the demands of CONOPs. Increasing the numbers of detectable, isolable, and recoverable failures would increase the mission assurance of the system.

In our example, a zero-trust analysis of the "survivable" mission computer is summarized as follows. For the hardware foundation, the incorporation of a hardware security module to implement secure boot provides a root-of-trust, but it is still

subject to zero-day hardware and firmware vulnerability. Also, there is no provision to verify the security module's behavior at operation. For the software environment including the separation kernel and virtual machine manager, both of them are verified for authenticity and integrity at boot time. The separation kernel is also formally verified at design time [2]. However, similar to the hardware layer, the software layer is subject to zero-day vulnerability and its behavior is not verified at operation.

The design has focused on defending software applications, which are verified for authenticity and integrity at boot time. Inter-process communications are controlled and monitored by the separation kernel. Communications are protected by cryptography. The application behavior is also monitored during operation. However, zero-day vulnerability remains a risk. Also, recovery is limited to reloading and restarting, which may not be sufficient to survive attacks without additional measures.

Apparently, the system stakeholder needs to take into account the significance of the system's residual vulnerability, and the return of investment of incorporating further features to meet survivability requirements. In many cases, the goal of a fully survivable system is non-feasible. The stakeholder, if needed, may modify the original CONOPS and mission success criteria to be "survivable."

At this point the system under design could be prototyped to measure its "real-time" performance with respect to its survivability requirements. In addition, overhead on system SWaP and performance should also be assessed. In general, a few iterations will be needed to arrive at a final design. Performance improvement is beyond the scope of this paper, but has been the subject of numerous resources, e.g., [1].

V. Summary and Ongoing Activities

The ZTA approach described in this paper has been successfully applied to the survivability development of mission-critical embedded systems including drones, microelectronics, and space communications. Ongoing research activities include integrating this ZTA approach with a system modeling and engineering tool, for example, MagicDraw [12].

VI. Acknowledgement

Fred Schneider is supported in part by Air Force Office of Scientific Research under award number FA9550-23-1-0435, as well as by funding from MIT Lincoln Laboratory, Amazon, and Google. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these organizations.

VII. References

[1] D. Martinez, R. Bond, and M. Vai, Ed., High Performance Embedded Computing Handbook, CRC Press, 2008.

[2] M. Vai, D. Whelihan, et al, Agile and Resilient Embedded Systems, MILCOM, October 2021.

[3] H. Whitman, M. Vai, et al, "HARDEN: A High Assurance Design Environment," GOMACTech, March 2019.

[4] D. Whelihan, M. Vai, et al, "Designing agility and resilience into embedded systems," MILCOM, October 2017.

[5] M. Vai, D. Whelihan, et al, "Systems Design of Cybersecurity in Embedded Systems," IEEE High Performance Embedded Computing Conference, September 2016.

[6] https://www.whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf, accessed June 27, 2023.

[7] Department of Defense (DoD) Zero Trust Reference Architecture, Version 2.0, July, 2022, Defense Information Systems Agency (DISA) and National Security Agency (NSA) Engineering Team, https://dodcio.defense.gov/Portals/0/Documents/Library/(U)ZT_RA_v2.0(U)_Sep22.pdf, accessed June 27, 2023

[8] https://www.ll.mit.edu/news/zero-trust-architecture-may-hold-answer-cybersecurity-insider-threats, accessed June 27, 2023.

[9] https://www.ll.mit.edu/sites/default/files/page/doc/2018-05/22_1_8_Okhravi.pdf, accessed June 20, 2023

[10] G. Klein et al., "seL4: Formal verification of an os kernel," *Proc. of the ACM SIGOPS 22nd SOSP'09*.

[11] D. Whelihan, M. Vai, et al, "SHAMROCK: AA SynthesizableSynthesizable High Assurance Cryptography and Key High Assurance Cryptography and Key Management Management Coprocessor," MILCOM, October 2016.

[12] https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/, accessed June 20, 2023.

[13] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf, accessed June 20, 2023.