WIND™

A SURVEY OF INFORMATION SECURITY IMPLEMENTATIONS FOR EMBEDDED SYSTEMS

By Arlen Baker, Principal Security Architect, Wind River

WHEN IT MATTERS, IT RUNS ON WIND RIVER

ABSTRACT

This paper examines the implementations of the well-known information security[1] components of confidentiality, integrity, and availability (the CIA triad[2]) as applied to embedded systems, and how these implementations can be used to defend against various attacks.

The approaches taken by this paper are widely applicable to embedded systems in a variety of markets, including aerospace,[3] automotive,[4] defense,[5] industrial,[6] medical,[7,8] and networking,[9] and are directly applicable to the protection of the intellectual property (IP) of the vendor.

The CIA Triad is authoritatively defined in:

United States Code, 2006 Edition, Supplement 5

Title 44 - Public Printing and Documents

Chapter 35 - Coordination of Federal Information Policy

Subchapter III - Information Security

Section 3542 - Definitions

TABLE OF CONTENTS

THE CIA TRIAD

The CIA triad is the foundational security principle for the protection of an asset. Its three components can be thought of as similar to the components of security for the contents of a home:

- **Confidentiality** is defined as maintaining the *privacy* of an asset. Solid doors, walls, and window coverings provide privacy for the contents of a residence.
- **Integrity** is defined as maintaining the *content* of the asset. An alarm system, a fence, and locks on the doors and windows maintain the integrity of a residence, such that the contents of the residence are kept intact.
- **Availability** is defined as the *accessibility* of the asset. The contents of the residence are available to the residents via passcodes to the alarm system and keys to the door locks.

The CIA triad can be further broken down into categories, which can then be broken down into implementations, as shown in Figure 1. The remainder of this paper will discuss these sub-principles and how each can be used to secure an embedded system.

Application of the CIA triad begins with the security assessment. The security assessment determines which CIA implementations are required based on vulnerabilities, risks, regulatory requirements, and IP protection needs, and balances those needs against cost, performance, and the operational environment. The security assessment will provide the Security Policy, which defines the security objectives for the embedded system: what the security-related events are, how they are to be constrained, when they are to be reported, and what actions to take in response to the events. The security assessment also provides the processes within the development cycle to assure that the security-related principles are implemented.

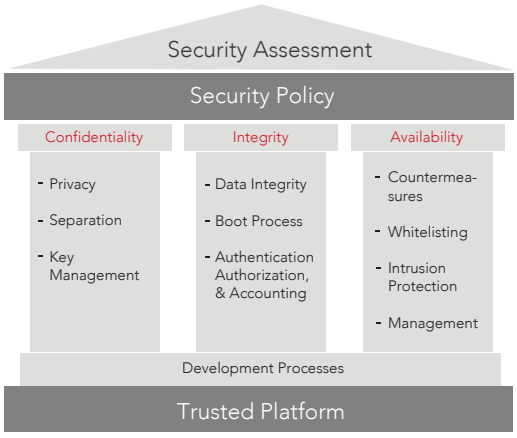

Figure 1. CIA triad principles

DEFENSE-IN-DEPTH APPROACH

No single security principle by itself can provide complete protection for an embedded system. Rather, it is the proper layering of these defenses that will provide a much stronger, multifaceted protection for the embedded system. The concept of layering these principles together is known as *defense in depth*.[10]

Many factors dictate the security components that need to be included to protect an embedded system; the security assessment will uncover the required components.

CONFIDENTIALITY FOR EMBEDDED SYSTEMS

Confidentiality implementations are used to protect the privacy of data in embedded systems. This protection includes data passing to/from the embedded system (data in motion), data that are stored on the embedded system such as on disk drives and/or in nonvolatile memory (data at rest), and data that are being processed on the embedded system (data in process). Confidentiality can be partitioned into three categories: privacy, separation, and key management (as shown in Figure 2 along with their associated implementations).

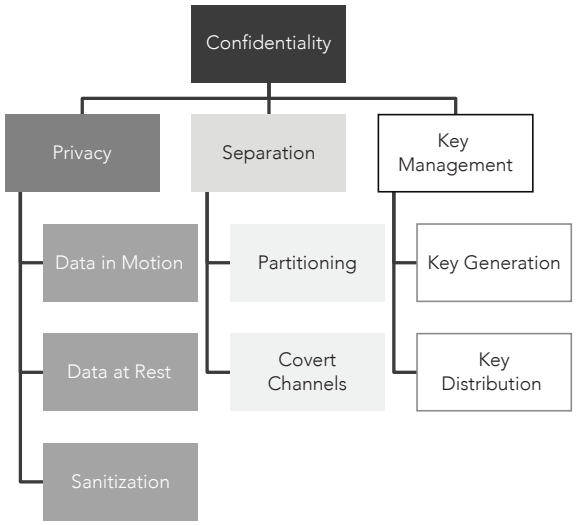

Figure 2. Confidentiality implementations

Privacy Implementations

Privacy is achieved through the use of cryptographic algorithms on the data (encryption), making it nonsensical to an unauthorized individual. An authorized individual can restore the data to its original form (decryption). Just as there are different types of door locks,[11] there are different types of cryptographic algorithms

based on the need. Types of cryptographic algorithms used for confidentiality include the following:[12]

- Symmetric algorithms
  - **Stream cipher:** Processing data one datum at a time
  - **Block cipher:** Processing data one group (multiple data) at a time
- Asymmetric (also known as public key) algorithms

The strength of the privacy provided is based on the combination of the cryptographic algorithm and the length of the associated key.[13] Industry-approved cryptographic algorithms and key lengths are provided in Barker, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths."[14]

The length of time in which a cryptographic key should be used is called a cryptoperiod. Cryptoperiods vary based on the algorithm, key length, usage environment, and volume of data that is being protected. Guidance for cryptoperiods can be found in Barker, "Recommedation for Key Management—Part1: General (Revision 4)."[15]

Symmetric cryptographic algorithms use the same key for both the encryption and decryption processing. This would be similar to a door lock that is keyed on both sides of the door for the same key: lock the door from the inside (encryption); unlock the door from the outside (decryption). An example of a symmetric algorithm is the Advanced Encryption Standard (AES).[16] AES is an industry-approved symmetric algorithm[17] for providing confidentiality of sensitive data. Figure 3 shows a typical symmetric cryptographic workflow. In the embedded arena, sharing a cryptographic key can be challenging because of the large number of end points involved. This challenge will be addressed in the "Key Management Implementations" section of this paper.

Asymmetric cryptographic algorithms are also called public key

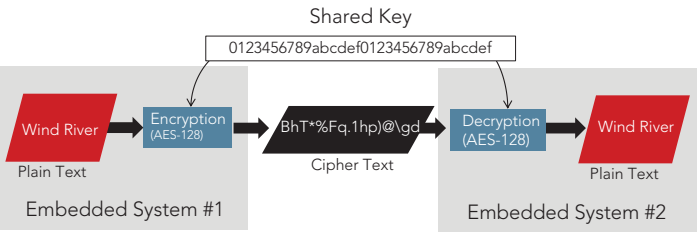

Figure 3. Symmetric cryptographic workflow

algorithms. This type of algorithm requires two keys (a key pair): one that is kept private and one that can be made public. The private key is kept tightly protected and is accessible by as few individuals as possible. The public key can be accessible by others, but does require a level of protection in an embedded environment, as its corruption could cause a denial-of-service (DoS) attack. The asymmetric algorithm provides for encryption to be completed by the public key and the decryption completed by the private key. Figure 4 presents an asymmetric cryptographic workflow.

A downside of asymmetric cryptography is that it requires more processing power and longer-length keys to achieve a level of security comparable to symmetric cryptography. For this reason, asymmetric cryptography is typically used for the generation and verification of digital signatures. This use will be discussed in the "Integrity for Embedded Systems" section of this paper.

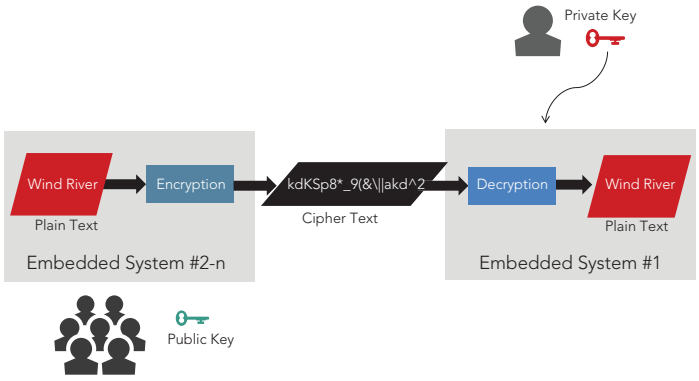

Figure 4. Asymmetric cryptographic workflow

Protecting the confidentiality of data in an embedded system can be a regulatory requirement, a method to protect IP, or an industry-recommended requirement (for example, in aerospace,[18] automotive,[19] defense,[20] industrial,[21] medical,[22] and networking[23]).

Data in an embedded system can be in one of three states: in motion, at rest, or in process. Data in motion is data passing to/from the embedded system; data in process is data generated or consumed within an embedded system; and data at rest is data stored on the embedded system.

Data-in-Motion Privacy

The data being passed over the network can be more than just the data being generated or consumed by an embedded system. The management data to and from the embedded system is just

as critical. Updates and patches, telemetry, and logging information can be of significant value to an attacker, so protection of this data is paramount to securing the embedded system. An attacker will monitor the behavior of the embedded system when stimuli are applied during attacks; by observing the response from the embedded system, the attacker can plan the next step in the attack process.

Many implementations for protecting the confidentiality of data in motion are available. These implementations can operate at different levels of the network stack—for example, Internet Protocol Security (IPsec),[24] Transport Layer Security (TLS),[25] and HyperText Transfer Protocol (HTTP) over TLS (HTTPS)[26]—and these just scratch the surface, as shown in Figure 5. Recommendations for securing data in motion are provided in McKay and Cooper, "Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations,"[27] using the TLS protocol.

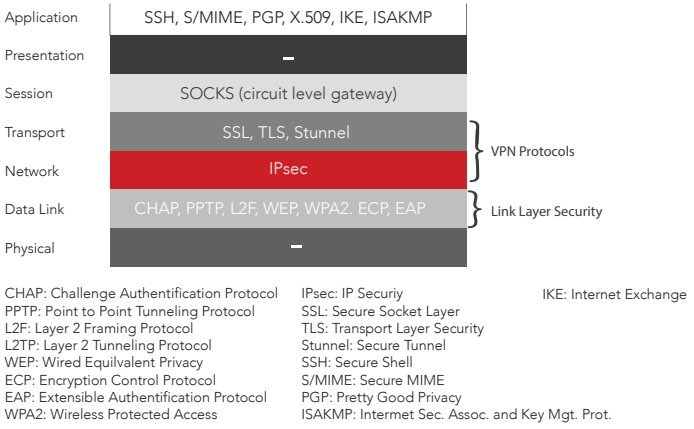

Figure 5. Security protocols and network layers[28]

Data-at-Rest Privacy

Protection of data stored on an embedded system, whether on a disk drive, a USB stick, or in nonvolatile RAM, can include the data produced or consumed by the embedded system, patches or other updates, telemetry data, IP, and logging information. These data can be very valuable to attackers if stolen, and if corrupted, can also disrupt the operation of the embedded system, causing a DoS attack.[29]

Protection of data at rest is typically implemented by the use of symmetric cryptography. Symmetric cryptographic algorithms such as AES are ideally suited for protecting data at rest, as they are fast and can accomplish strong levels of protection using shorter key lengths than asymmetric cryptography. The symmetric key, however, must be tightly protected on the device to maintain the privacy of the data. Secure storage of the symmetric key is best accomplished using hardware assistance such as the Trusted Platform Module[30] or the Secure Key Management Module of an NXP (formerly Freescale) QorIQ processor.[31] If hardware assistance is not available for key storage, a software implementation can then be used that utilizes obfuscation to achieve its protection.

The usage environment of the embedded system may have an impact on the generation of the key. For example, if the data need to be recovered and the embedded system becomes inoperable, the ability to have a duplicate key is then required. Conversely, if the data are isolated to the embedded system and do not need to be recovered, the key can be randomly generated and stored solely on the embedded system in a hardware-assisted, protected mechanism.

Sanitization

When critical data are no longer needed on the embedded system, they should be sanitized to prevent loss of privacy. To minimize the potential attack surface, the data's memory locations should be overwritten, including the stack. The simplest method of keeping the stack sanitized is to initialize all local variables within a function when the variables are defined. As the functions are executed, the stack is auto-sanitized as a result.

Because the use of cryptographic algorithms inherently requires the generation and use of a cryptographic key, the means to destroy or erase this key and associated data is required. By encrypting all data at rest on the embedded system, sanitization can be completed by the destruction of the cryptographic key(s) used to protect that data. Once the key is destroyed, the data cannot be restored and will remain nonsensical. This approach is called cryptographic erase.[32] Destruction of the cryptographic key(s) is based on the hardware device, if hardware is used. If not, a series of overwrites of alternating data patterns over the key storage area can be used.

Separation Implementations

The architecture of the embedded system provides a level of confidentiality by keeping independent functions isolated from each other. For example, the function that provides connectivity to the Internet can be kept separate from the function that accesses the sensors the embedded system is managing, with strict informa-

tion flows between the two. This constrains an attack on the Internet connectivity function from impacting the sensor management function.

Partitioning (Data in Process)

Isolating processes on the system provides a level of confidentiality by assuring that each process cannot access or interfere with other processes. A typical implementation of separation uses a Type 1 hypervisor that resides directly above the processor. The hypervisor defines partitions by assigning resources (memory, devices, cores, etc.) to each partition, and executes the process(es) within each partition. The processes can execute within the defined partition and maintain the privacy of its data from other processes on the processor.

Communication channels between the partitions can be provided by the hypervisor to include shared memory regions or an inter-partition communication mechanism. These communication channels can be restricted in terms of direction of flow, access rights, and whether or not the hypervisor itself is involved in the communication flow.

The assignment of resources and definition of the communication channels is defined in the hypervisor's Security Policy. The Security Policy creates the time and space allocation for each partition along with the channels of communication, if any. This communication policy can be as high level as "yes" or "no," or granular enough to constrain the size of the message, the data rate, and the direction of the communication path.

Covert Channels

Care must be taken in the implementation and usage of communication channels. An unintended path of communication could be created, exposing critical data to an attacker. These unintended communication paths are called covert channels and come in two forms: covert storage channels and covert timing channels. A covert storage channel is created when data are passed against the intent of the Security Policy. An example of this is using the IP packet header to transmit data.[33] A covert timing channel is created when the occurrence of an event over a period of time passes information.[34]

Covert channels can exist in all types of computing environments, from within a partitioned system to a typical embedded system.[35] When strong security is required, an analysis should be performed to identify and remove or mitigate channels that could be used by an attacker.

Key Management Implementations

Coordinating the creation, use, delivery, and updating of cryptographic keys is called key management. The complexity of key management is based on the number of keys, the number of systems that require those keys, and the cryptoperiod of each key.

In the simplest but least secure form of key management, all embedded systems use the same key for an infinite amount of time (i.e., non-expiring cryptoperiod). This configuration provides for a level of privacy by using cryptography (with the assumption it is correctly applied), but has the risk of exposing all embedded systems to attack if the key is ever compromised. In contrast, a configuration where each embedded system has its own unique key for a short, random cryptoperiod leads to much stronger security.

The complexity of ensuring that the many systems in a typical embedded environment have the correct cryptographic keys and maintain their recommended cryptoperiods demands a central key management system that implements the framework specified in Barker, Smid, Branstad, and Chokhani, "A Framework for Designing Cryptographic Key Management Systems."[36]

Key Generation

Random numbers are a cornerstone to the effectiveness of cryptography. Pseudo-random numbers (PRN) are a starting point, but they do not provide the strength of a true (entropic) random number (TRN) generator. The issue with PRNs is that they are deterministic by definition, as they are generated by a mathematical equation. Determinism in cryptographic key generation is counterproductive because if the starting point of the PRN (the "seed") is found, the cryptographic key can be determined in a much shorter timeframe.

Because of the shortcomings of a PRN, a hardware-based random number generator that provides true entropy can be used for the generation of strong cryptographic keys. Modern Intel® processors have specific instructions[37] available and the NXP processors have a hardware-based random number generator available in their SEC module.[38] With this hardware support, a platform is available such that cryptographic keys can always be created using a TRN generator.[39] If hardware support is not available, there are software-based alternatives that can be used to ensure a quality source of entropy.

Key Distribution

Extreme care is required when distributing cryptographic keys to embedded systems. Verification of the destination embedded system must be absolute to ensure the keying material does not mistakenly fall into an attacker's hands. The keying material must be kept confidential, from the source to the destination embedded system and within the embedded system, until it is needed. The embedded system must have absolute verification of the source of the key distribution to ensure an attack is not being launched against it. By using an attacker's keying material, the embedded system unknowingly loses the confidentiality of its data in motion.

INTEGRITY FOR EMBEDDED SYSTEMS

Integrity implementations are used to ensure that the data of the embedded system have not been modified or deleted by an attacker. These data include the data being generated or consumed by the embedded system, along with its programming data (operating system, applications, configuration data, etc.). As with confidentiality, integrity applies to the three states in which data exist: in motion, at rest, and in process.

Integrity of data is typically verified by a mathematical algorithm called a hash. There are many implementations of hash functions, but the one selected must minimize, if not completely remove, the risk of a collision where more than one input can generate the same output (the message digest). These types of hashes are called Secure Hash Algorithms (SHA).[40] By minimizing the chance that the changed data hashes to the same value as the original data, this removal of collision risk increases the probability of a change to the data being verified. As shown in Figure 6, the categories of integrity for an embedded system are: data integrity, boot process, and AAA.

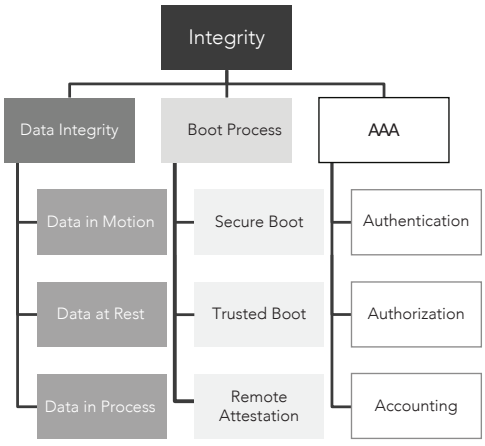

Figure 6. Integrity implementations

Data Integrity Implementations

The data within an embedded system can include the operating system, the applications, and the configuration data. If any of these data are corrupted, the embedded system will not perform its intended function, or the embedded system can become an instrument for the attacker to use on the fabric of the embedded system itself (also known as a Bot). Disruption of the integrity of the data on the embedded system is like disruption of the foundation of a residence: the components above it are weakened and cannot be trusted.

Data-in-Motion Integrity

Many embedded systems have the requirement of passing data to/from the device. The data must be protected against modification, either intentional (through an attack) or unintentional (through a programming error), while being transmitted from source to destination. A hash can be used, but the attacker can circumvent this measure by simply recalculating a new message digest after the modification has been made. A stronger integrity mechanism is a keyed-hash message authentication code (HMAC). The HMAC provides a data integrity check with a shared private key, as shown in Figure 7. Because the HMAC requires a key, it must be protected just like a cryptographic key.

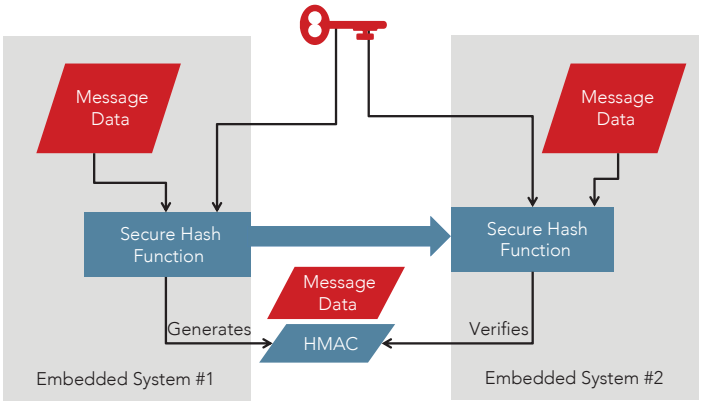

Figure 7. HMAC workflow

Data-at-Rest Integrity

The integrity of critical data on the embedded system must be verified before they can be relied upon for processing. The verification of the programming data will be covered in the "Secure Boot" section of this paper. Verification of the configuration and site-specific data should be completed prior to operating on that data to ensure no modifications to these data have been made by an attacker. Using an HMAC, the message digest of the data can be calculated (periodically and before shutdown) and verified (at startup and periodically).

Data-in-Process Integrity

As data are being generated or consumed on the embedded system, integrity checks can be used to ensure that the processing flow can be trusted and that the correct data are being processed. Unique enumeration values throughout the software and for all data types can be used to verify the processing flow integrity. This approach ensures that each application programming interface (API) is called with the parameters that are unique to only that API, and thus maintains the integrity of the processing flow.

Boot Process Implementations

Starting the embedded system with known, authenticated software is foundational to securing the embedded system. Without a boot process that proves the embedded system is starting with unmodified software and data, the system cannot be trusted. The verification must include boot code, application code, and critical data that are stored on the system (data at rest).

The boot process is partitioned into two parts, as shown in Figure 8. The Secure Boot phase is controlled by the trusted hardware platform, and the Trusted Boot phase is controlled by the previously verified software.

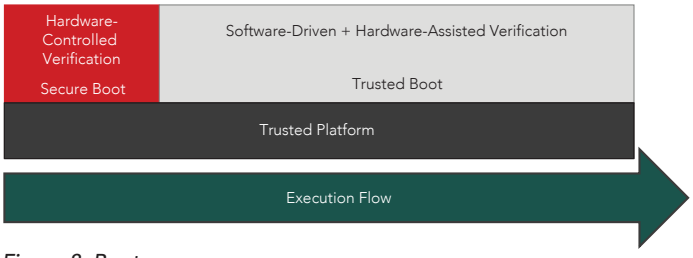

Figure 8. Boot process

Secure Boot

Secure Boot starts with the platform and the pedigree of that platform. A Trusted Platform is hardware that has been purchased through approved channels of distribution by the technology supplier.[41] Upon receipt of the platform, it must be validated that the correct item was received, its delivery path is sensible, its delivery time is justified, and its tamper-resistant packaging is intact. Only then can the platform become a Trusted Platform.

One of the security features that the Trusted Platform must provide for is a mechanism to verify the boot software of the embedded system. This mechanism enables an unchangeable (due to its implementation in hardware) process to verify the first piece of software in the boot process. The verification process is best implemented using digital signatures.

Digital signatures combine the use of a hash function's message digest along with the asymmetric private-key encryption of that hash function by the author. The Trusted Platform verifies that digital signature by recalculating the message digest, decrypting the associated digital signature with the public key, and comparing the message digests. If the message digests match, then the integrity of the software is verified. This workflow is shown in Figure 9, and differs from what is shown in Figure 4, as that workflow was for confidentiality and this workflow is for integrity.

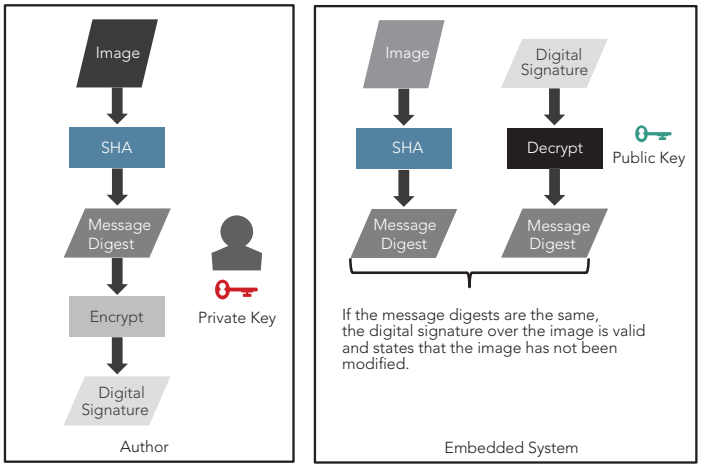

Figure 9. Digital signature workflow

Trusted Boot

Trusted Boot is the progression of a boot process where individual images and data are verified by previously verified software. It is best if this process includes hardware assist to perform the verification processing, because the immutable properties of hardware (whether system-on-chip [SoC] or field programmable gate array [FPGA]) mitigates the risk of a malicious change causing a breach in verified boot processing. The process of one verified image passing control to another verified image is called a chain of trust (see Figure 10). The chain-of-trust approach ensures that only verified software is loaded into the system.

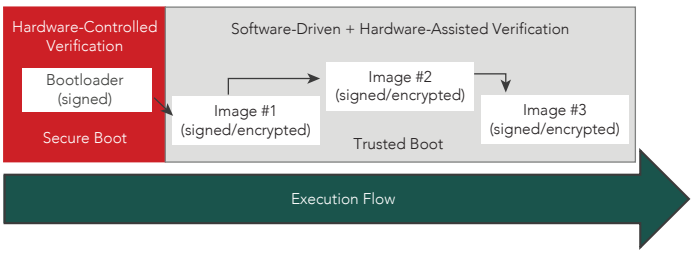

Figure 10. Chain of trust

Remote Attestation

Remote attestation[42] is the process of taking "measurements" during the Secure Boot and Trusted Boot phases of the boot sequence and reporting these measurements to a physically separate server, as shown in Figure 11. These measurements are typically a hash of the components of the boot process (bootloader, applications, etc.). The server than compares these measurements and determines the trustworthiness of the embedded system. The transmission of the measurements is secured and must include the identity of the embedded system. This identity is most secure if it is a hardware-bound identity, such as found in a Trusted Platform Module (TPM). The baseline measurements must be made in a trusted environment and by trusted individual(s).

Remote attestation is best for an embedded system that is "always on" a network connection (rather than one that has limited network connectivity) so that a mismatch in the measurements can be quickly identified, and the response defined in the Security Policy can be enacted.

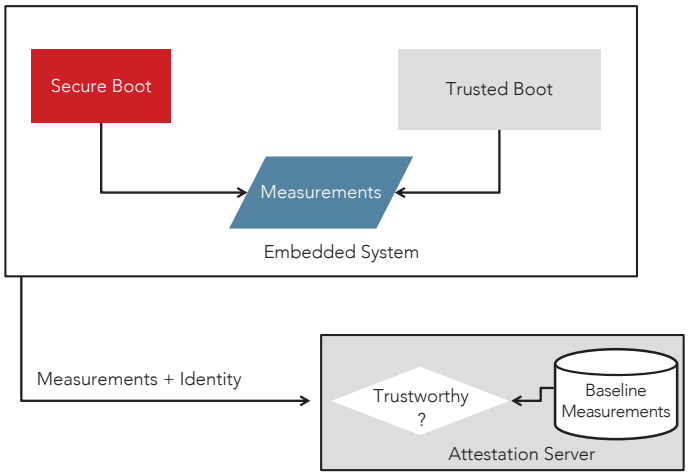

Figure 11. Remote attestation workflow

Authentication/Authorization/ Accounting (AAA) Implementations

Security risks increase as the level of exposure of an embedded system to the Internet increases. Couple exposure to the Internet with a network configuration that is dynamic, and the embedded system requires a layering of defenses to maintain the integrity of its knowledge of the other devices it communicates with. This layering of integrity defenses is called authentication, authorization, and accounting (AAA, pronounced "triple A"). AAA provides a level of integrity for the embedded system in determining which other devices on its network it is allowed to communicate with and the type(s) of data that should be passed.

Due to the nature and complexity of these defenses, a centralized server and toolset are required to properly manage the application of these defenses in an embedded systems environment and to analyze and respond to any received security events.

Authentication

Authentication tries to answer the question: "Are you who you say you are?" in order to establish trust in the identity of the distant device the embedded system is attempting to communicate with over the network. To implement authentication, a trusted third party is required. This trusted third party mediates between the embedded system and the distant device (so both devices must be known to the trusted third party) and contains information about all devices on the network. A well-established protocol called Kerberos[43] exists to establish this trust between devices over the network. Using the Kerberos-identified messages along with clarifying annotations, Figure 12 shows the message flow for the embedded system to establish its identity so it can communicate with the distant device over the network.

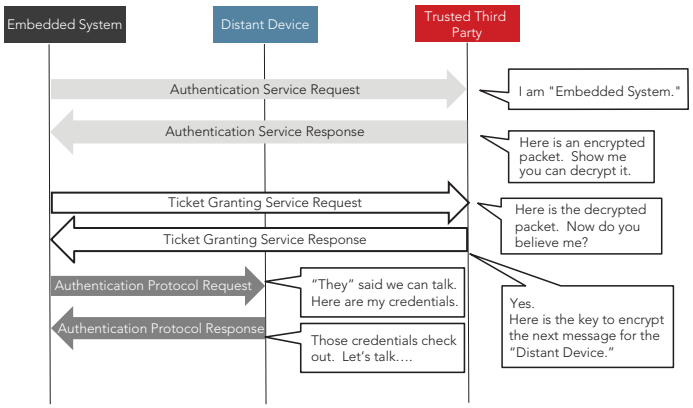

Figure 12. Annotated Kerberos message flow

Authentication can also occur within the embedded system with its applications and operating system. In this case, the trusted third party consists of both the operating system and the Security

Policy, which will have a set of identifying attributes unique to each application (whether a task, process, or partition). The operating system will have access to these immutable identifying attributes at runtime. When the application makes a service request to the operating system, the request includes these identifying attributes (without the awareness of the application). The operating system then has assurance of the source of the request prior to providing the service.

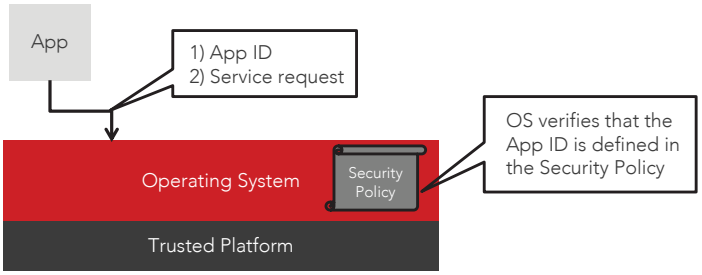

Figure 13. Authentication provided by a robust operating system

Authorization

After authentication, authorization typically follows, because the embedded system must be known before it is allowed to access other network resources. Authorization is the determination of the type of access allowed to a resource within the network. A network-wide Security Policy defines what the resources are, the paths of communication to and from each resource, and to what level the access can occur (read versus read/write).

Just as with authentication, authorization can occur *within* the embedded system as well. The Security Policy being enforced by the robust operating system is used to make the determination about whether the application is allowed to request the service. This is the second step shown in Figure 14.

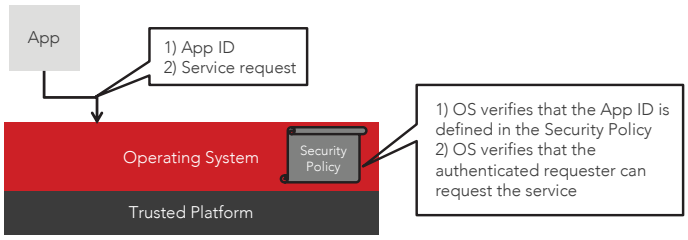

Figure 14. Authorization of a request provided by a robust operating system

Accounting

Accounting is the generation of a log of events that denote security-related activities on and by the embedded system. This logging of events can occur within the embedded system or to an external server. The events to log and the response to those events are defined by the Security Policy.

Because impenetrable defenses are not possible, the ability to determine what led to an attack and what happened during the attack are critical in the feedback loop for improving the security of the embedded system. Security-related event logs must be collected and analyzed to ensure the following:

1. The integrity of the embedded system can be monitored
2. An attack determination can be made
3. A response to the attack is made

It is best that these occur as near to real time as possible to minimize the damage from the attack. The Security Policy on the embedded system defines what the security-related events are and the initial, front-line response to that event.

Because of the large number of embedded systems that must be monitored, along with the large number of events that need to be parsed and managed, a specialized server is required. This type of server is called a security information and event management (SIEM) server. A SIEM server is able to correlate received security event messages from embedded systems and use predictive analytics to determine if an embedded system is at risk of an attack. A SIEM server can also take as an input a continuous threat intelligence feed to aid in its analysis of the security events. This intelligence feed provides a larger view of the type of attacks that the embedded system may face.

A SIEM server is a powerful tool but also a very complex tool. The introduction of a SIEM server into an infrastructure must be carefully planned, and to be successful its capabilities must be allowed to evolve over time, as the administrators become more accustomed to what it can provide.

A notional configuration of using a SIEM server is shown in Figure 15.

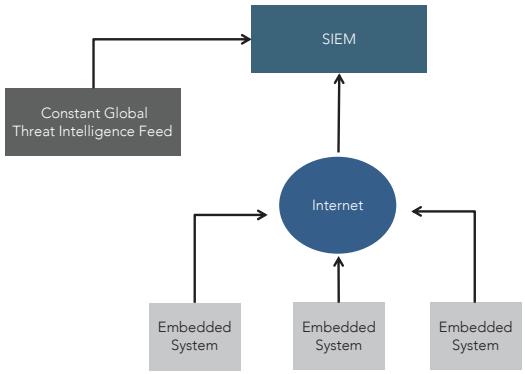

Figure 15. Notional SIEM server usage

AVAILABILITY FOR EMBEDDED SYSTEMS

Availability implementations are used to ensure that an embedded system performs its intended function. The simplest and purest approach to ensuring availability for an embedded system is never to allow any changes to occur on the embedded system. Of course, this is unrealistic, because attacks over the Internet constantly evolve in sophistication, and the functionality of the embedded system itself evolves over time. Because of these competing goals (availability vs. enhancements), a series of implementations is required to reduce the risk of diminishing the availability of the embedded system. As shown in Figure 16, the categories of availability for an embedded system are: whitelisting, intrusion protection, management, and countermeasures.

Whitelisting Implementations

Whitelisting simply defines what is allowed. Anything that is not on the whitelist is denied. Whitelisting for an embedded system can be applied at the network level (what devices the embedded system can communicate with) and within the embedded system (what applications can be executed within the embedded system). The focus in this section will be within the embedded system, on what can be layered to maintain the availability of the embedded system.

Access Control

Defining which user or application can execute a specific service provided by the embedded system provides a level of defense against an attack. To correctly determine which user or application is associated with an exploitable service of the embedded system, the attacker would need an almost *a priori* knowledge of the embedded system (and why the accounting implementation is required to detect this "guessing"). This user/application-to-service control is called *access control*. Access control is the policy that defines what a user or application can perform with a service the embedded system is providing. There are four different types of access controls as listed in Table 1, which also describes the management overhead of implementing the access control and the strength of access it provides (assuming the access control is non-bypassable).

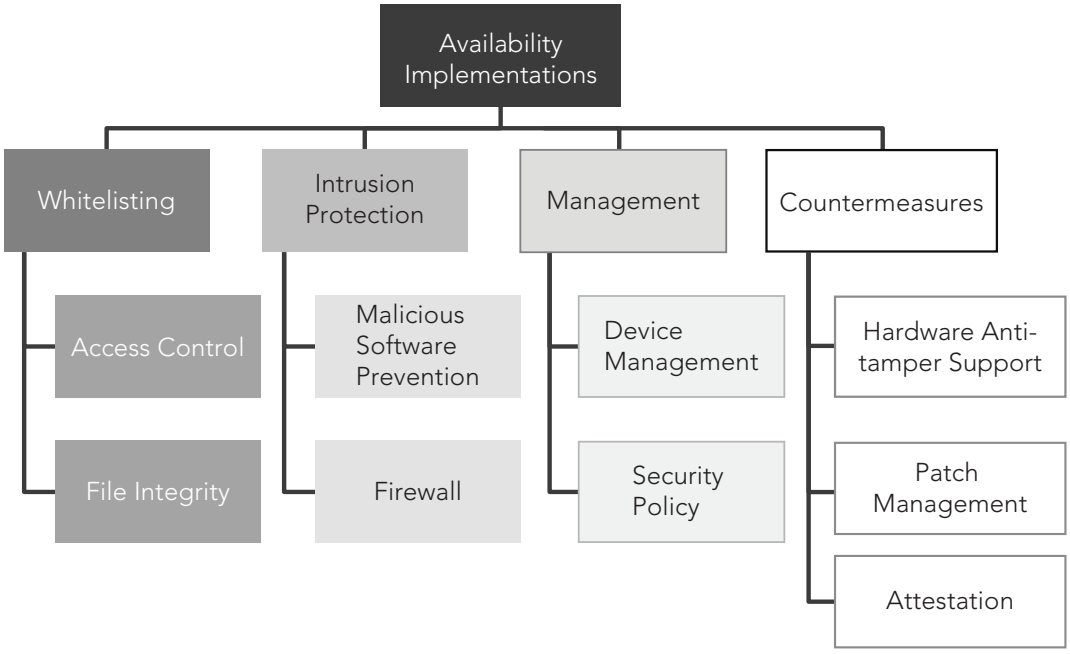

Figure 16. Availability implementations

Table 1. Access Control Types

Access Control Type	Definition	Example	Overhead	Strength
Mandatory access control (MAC)	Each service is labeled and each user/application is labeled. Access is allowed only if the labels match.	"Only the maintenance application can access the reboot service of the embedded system."	Very high The labeling between the users/applications and services must be defined and maintained.	Strongest Clearly defines what user/application can access what service.
Role-based access control (RBAC)	The access controls are specific to the *role* that the user/application is providing.	"Only the network administrator application can update the AAA configuration."	Medium Once roles and services are defined, it becomes more of a maintenance task to keep them updated.	Medium-to-high Based on the granularity of the roles defined.
Discretionary access control (DAC)	The access controls are specific to the user/application and defined by the user/application.	"The file created by the Sensor Application can be read, but not written, by the Network Application."	Very Low It is up to the application to define the access control.	Weakest Pushes determination to the application.
Rule-based access control (RBAC)	The access controls are based on a set of rules which are more than just the user/application.	"Access to maintenance data is limited to 2300-2359 each night and only when the embedded system is at a maintenance facility."	Medium-to-High Depends upon how fluid the rules are and if there are exception cases.	Medium Depends on the stringency of the rules.

File Integrity

Maintaining the integrity of the data files that are generated or consumed on the embedded system aid its availability by not allowing any corrupt or malicious data onto the device and not allowing a polluted file to pass through the embedded system. Depending on the environment in which the embedded system operates, how the embedded system is implemented, and the regulations with which the embedded system must be compliant, the integrity of the files and the list of files on the device must be closely monitored. Otherwise, the embedded system can be attacked and lose its availability.

Steganography is a form of covert channel where data are hidden in an unused portion of a file and are only evident by the antagonist and the intended recipient. These hidden data can include any type of confidential or protected data, or even malware.

Borrowing from the integrity principle, a "seal" can be applied on the files as close to the source as possible, and as soon as the file is created, to provide the highest level of assurance that it has not been tampered with. It is best if the integrity seal is a digital signature, where the source has the closely guarded private key and the embedded system has the public key. The integrity seal is then verified by the embedded system as part of the boot process and before each access (i.e., read, execute, and pass-through [as needed]).

Intrusion Protection Implementations

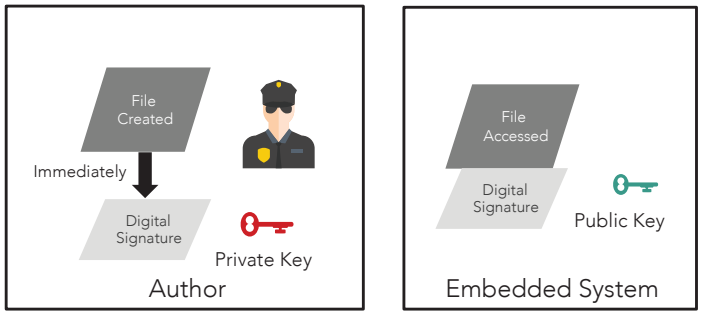

Figure 17. File integrity workflow

Intrusion detection systems (IDS) and intrusion prevention systems (IPS) are common systems in the topology of a modern network. The terms can be confusing and overloaded, but Google searches yield the following definitions:

- **Detect:** To discover or identify the presence or existence of
- **Prevent:** To keep (something) from happening or arising
- **Protect:** To keep safe from harm or injury

So protection of the embedded system against malicious software will be defined as:

```
protection :== detection + prevention
```

Intrusion protection is specific to an embedded system, depending on both its environment and its functionality. Intrusion protection provides a layer of defense within the embedded system to detect the presence of, and prevent damage from, malicious software executing within the embedded system's memory space. Intrusion can occur through the network or through a separate device connected to the embedded system, such as a USB device.

Malicious Software Prevention

Preventing foreign software from executing within the embedded system requires a static inventory of what should be on the embedded system, along with a known list of APIs that each application within the embedded system can access. This inventory includes the memory, the file system (if applicable), and the critical system APIs that are allowed to be called by each application.

The inventory of the file system and memory is protected, verified at startup, and periodically verified as the embedded system executes. The inventory needs to be defined in such a way that a straightforward verification process can occur (to minimize performance impacts). It is best if the unused memory within the system can be periodically verified to ensure no malicious software has infiltrated the embedded system.

Applications should never be given complete access to all APIs available by the operating system. Rather, an enforced subset of the APIs that the application absolutely requires to fulfill its requirements should be allocated. This is the principle of least privilege, which states that an entity should be restricted in access to only those resources required to fulfill its function, and no more. For example, an application that monitors a sensor does not require access to APIs that control the execution model of the system. For better security, the enforcement of the APIs should be implemented by the OS against a statically defined configuration determined at build time.

As shown in Figure 18, protection against malicious software requires both the ability to detect and a mechanism to prevent an attack from the malicious software.

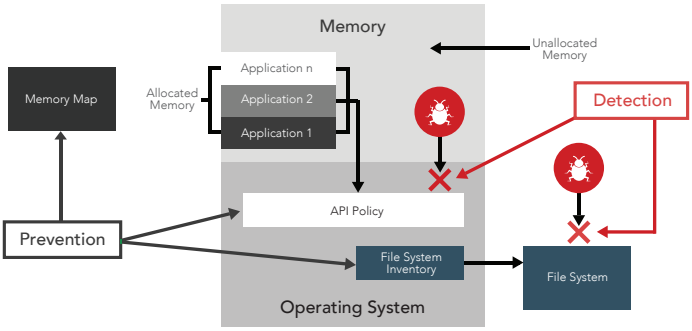

Figure 18. Malicious software protection

Firewall

A firewall is defined as a system that monitors and controls the incoming and outgoing network traffic based on predetermined security rules.[44] For an embedded system, it is best if the firewall rules are partitioned into different layers to simplify the management of the rules, and more importantly, to distribute the rules across multiple applications and files to decrease the amount of damage that can be done by an attack. This layering of rules is shown in Figure 19, which shows how different applications modify the different firewall rules.

It is important that a firewall be able to determine anomalous behavior for communications ports. Each port into and out of the embedded system should be characterized to determine its data rate (either sustained or burst). The firewall should then contain rules to monitor the rate of data going through each port. This characterization can then be used to help in detecting an intrusion into the embedded system.

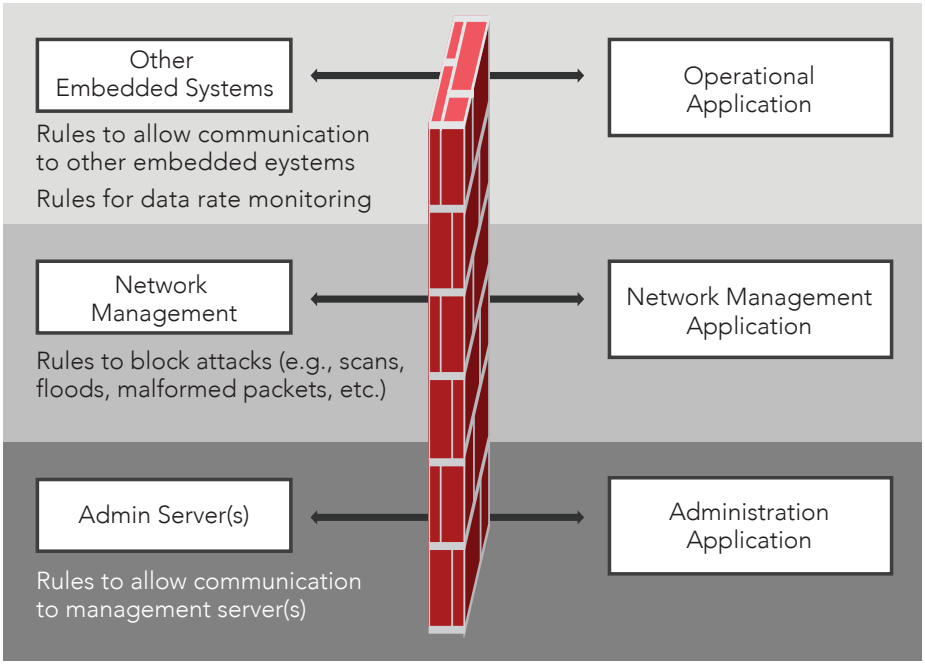

Figure 19. Firewall rule layering (conceptual)

Embedded System Management Implementations

Since the embedded system connects to the Internet, it must operate in a very dynamic and ever-growing cyberattack[45] environment. A set-and-forget management approach will protect the embedded system for only a short period of time. Rather, active management of the device is required to maintain its availability to perform its function. As with AAA, a centralized server and toolset are required to provide the level of management of a large number of embedded systems.

Device Management

Communication between the embedded system and its centralized server must be the most protected and most layered communication within the device. Because of the administrative commands that drive the embedded system, this communication path is the most sought-after by an attacker. Faux-commands at random intervals should be considered for several reasons. It will be difficult for the attacker to understand a pattern of communication, to determine if it is "real" or not, and to determine if it is due to a stimulus applied to the embedded system. These faux-commands should vary in size from a typical "real" command to a patch update and everything in between.

The embedded system requires different levels of management to maintain its availability. Overhead-type activities such as provisioning, commissioning, and general remote administration are typically required.

The next level of management involves the authentication and authorization described in the AAA section of this paper. It is mandatory that the embedded system is not allowed to communicate with any device outside of its whitelist of approved devices, and that accounting entries are received when the communication is occurring.

The highest level of management is that of security-related management. This includes offloading security event logs, critical security events, and security-related patch updates.

Security Policy

The Security Policy describes what the embedded system needs to protect, how to protect it, and the events related to that protection. The Security Policy covers the implementations covered in this paper. Examples of items covered by the Security Policy are listed in Table 2. Protection of the Security Policy on the embedded system demands, at minimum, an integrity check to ensure it has not been corrupted. The embedded system will need to have a preprogrammed response to a corrupt Security Policy (destruction of nonvolatile files, shutdown, etc.).

Table 2. Security Policy Examples

Triad	Implementation	Security Policy Examples
Confidentiality	Privacy	Algorithm(s) to use for each path of communication and each data stored
	Separation	The partitions in the system; information flow between those partitions; the devices accessible by the partitions; the OS APIs allowed to be called
	Key management	The list of crypto keys and their cryptoperiod; how each key is protected and its destruction mechanism
Integrity	Data integrity	Which integrity mechanisms to use on which data if applicable: the HMAC key and its cryptoperiod
	Boot process	What needs to be verified; what if a verification step fails (degraded mode)
	AAA	What communication needs to be authenticated; what is authorized to be accessed on the device; security events that are to be logged locally and sent remotely
Availability	Whitelisting	What access control method is used; list of resources on the device that require controlled access
	Intrusion protection	APIs allowed to be executed by the applications; the file system inventory
	Management	Management server information; communication path implementation; rate and type of faux-commands
	Countermeasures	Response to tamper events and attestation violations

Countermeasure Implementations

Availability can be larger than the single embedded system, but can include a much larger group of embedded systems. The compromise of one embedded system can lead to a compromise of all embedded systems. Because of this, specific defenses are required to minimize an attack on the larger group of embedded systems.

Hardware Anti-tamper Support

If an attacker has physical access to the embedded system, internal components of the device become a priority for the attacker, up to and including attaching an external device to the embedded system (e.g., a JTAG device). In general, embedded systems should be encased and only necessary ports exposed. But if an attacker opens the case of the embedded system and has access to its components, great damage can occur not only to the device, but to the network that the embedded system resides on and the devices that it communicates with. A defensive layer against physical access to the embedded system can be implemented using anti-tamper[46] means provided by the Trusted Platform.

The embedded system's Security Policy defines what is to be protected from a physical attack, but considerations should include encrypting applications and configuration data so that the cryptographic key(s) is erased when a physical tamper event occurs (the sanitization implementation). This approach is outside the function of software and is completely implemented by the Trusted Platform. When the system attempts to boot, the software will not be able to be decrypted correctly, rendering its contents unusable by the attacker. The same applies for the configuration data (and other data) on the embedded system. Without the cryptographic key, the data on the file system is unreadable by the attacker.

Although the attacked embedded system is no longer available, the remaining embedded systems and network continue to remain available.

Patch Management

A centralized server with a specialized toolset is required to manage the different patches on each embedded system. Because the embedded systems can reside in a large range of operational environments, some devices may require patches that other devices do not. These operational environments may also dictate a specific Security Policy that defines responses to certain attacks.

Attestation

Portions of the embedded systems application are typically static during its operation. The application itself, its operating system, and its configuration data are typically static. These regions of the embedded system can be verified during execution with assistance from the Trusted Platform. Following the boot process, an integrity calculation can be made over these regions, and then subsequent integrity calculations can be made and verified

against the boot-time integrity calculation by the Trusted Platform until the following power cycle. If there is a mismatch with the boot-time integrity calculation, the Security Policy will state what the response should be, including a system restart to cause the embedded system to go through its boot device.

A change in the static region of the embedded system could indicate a programming error, or that an attack is occurring and attempting to disrupt the operation of the device. Using the Trusted Platform, the performance impacts of this implementation can be significantly minimized.

CASE STUDIES—HOW TO APPLY THE CIA TRIAD

This section provides a notional representation of how to map a security specification to the implementations of the CIA triad defined in this paper. A security assessment is required to fully capture all requirements for a compliant solution (thus no warranty or claims are made on the solutions presented here), but this section illustrates how the implementations of the CIA triad are used to define a Wind River®–based solution.

Case Study: Aerospace Market

This case study shows a mapping of the recommendations from DO-355 "Information Security Guidance for Continuing Airworthiness." Note that this document has three separate sections for the Airborne Platform (sections 2 and 3), the Access Points (section 4), and the Ground Station Equipment (sections 5, 6, and 7). Table 3, Table 4, and Table 5 provide the mapping of DO-355 to the CIA triad implementations.

Table 3. Aerospace Case Study—Confidentiality Mapping

Confidentiality						
Separation		Privacy			Key Management	
Partitioning	Convert Channels	Data in Motion	Sanitization	Data at Rest	Key Distribution	Key Generation
		2.1 2.2.8 6.2.1	2.2.8 3.2.5 5.2.1 6.2.6 8.2.5	2.1	2.2.3 7.1 7.1	2.2.3 7.1 7.1

Table 4. Aerospace Case Study—Integrity Mapping

Integrity								
Boot Process			Data Integrity			AAA		
Trusted Boot	Secure Boot	Remote Attestation	Data at Rest	Data in Process	Data in Motion	Authentication	Authorization	Accounting
5.2.1 6.2.4	5.2.1 6.2.4 7.1		2.1 2.2.4 2.2.7		2.1 2.2.1 2.2.4 2.2.6 6.2.1	2.2.1 4.1 4.2 7.1	4.1	4.2 6.2.7 8.1

Table 5. Aerospace Case Study—Availability Mapping

Availability								
Whitelisting		Intrusion Protection		Managemet		Countermeasures		
Access Control	File Integrity	Malicious Software Prevention	Firewall	Device Management	Security Policy	H/W Anti-Tamper Support	Patch Management	Attestation
5.2.2 5.2.3 6.2.2		2.3.4	5.2.2 5.2.3 6.2.3 6.2.4	5.2.2 8.1 8.2.5	5.2.2 6.2.3 8.1 8.2.2.2 8.2.3	5.2.1 8.1	5.2.1 6.2.4 8.1 8.2.5	5.2.1 6.2.4 8.1

Using this mapping and comparing it to the Wind River offerings for confidentiality, integrity, and availability, a notional multi-OS solution can be constructed as shown in Figure 20 as a representative example to fulfill these requirements.

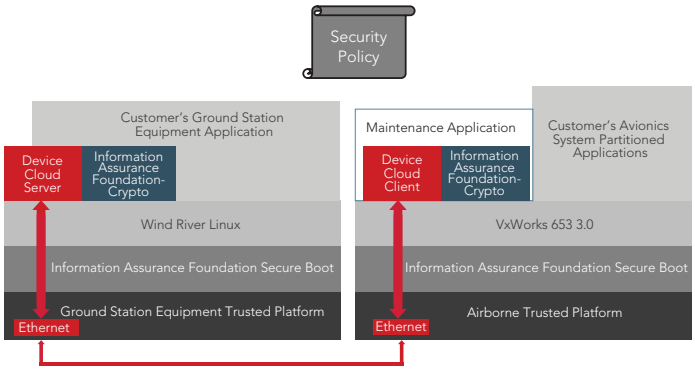

Figure 20. Aerospace case study—a notional approach

Case Study: Medical Market

This case study shows a mapping of the Security Capabilities Section of the *Manufacturer Disclosure Statement for Medical Device Security*[47] to the components of the CIA triad shown in Table 6, Table 7, and Table 8. Note that n-x denotes that all categories within section "n" map to that implementation.

Table 6. Medical Case Study—Confidentiality Mapping

Confidentiality						
Separation		Privacy			Key Management	
Partitioning	Convert Channels	Data in Motion	Sanitization	Data at Rest	Key Distribution	Key Generation
		7-x 18-2	6-x 16-2	17-1		

Table 7. Medical Case Study—Integrity Mapping

Integrity								
Boot Process			Data Integrity			AAA		
Trusted Boot	Secure Boot	Remote Attestation	Data at Rest	Data in Process	Data in Motion	Authentication	Authorization	Accounting
15-2 15-10	15-2 15-10		9-x 15-2 19-1		7-x 9-x 19-1	1-1 11-x	3-x	2 2-1 2-2.x 2-3.x 10-1.2

Table 8. Medical Case Study—Availability Mapping

Availability								
Whitelisting		Intrusion Protection		Management		Countermeasures		
Trusted Boot	Secure Boot	Malicious Software Prevention	Firewall	Device Management	Security Policy	H/W Anti-Tamper Support	Patch Management	Attestation
3-x 8-x 10-1.3 15-4 15-5 20-2	10-1 15-4 15-9	10-1 15-9	1-1.1 11-x 15-7 15-8 18-3 20-2	10-1.1 10-2 10-3 15-11	3-x 4-x 10-1.1 15-5 15-6 15-9 15-11 20-2 21-2.1	13-x	5-x	10-1.2

Using this mapping and comparing it to the Wind River offerings for confidentiality, integrity, and availability, a notional Linux-based[48] solution can be constructed as shown in Figure 21 as a representative example to fulfill these requirements.

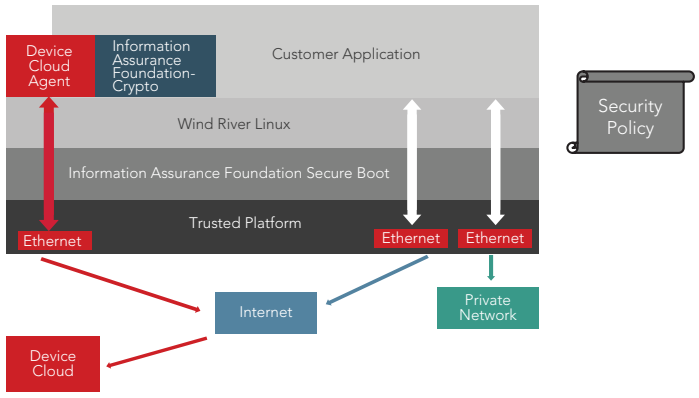

Figure 21. Medical case study—a notional approach

REFERENCES

Note: All web links accessed 06 February 2020

1 "Information Security," [Online]. Available: https://en.wikipedia.org/wiki/Information_security.

2 C. Perrin, "The CIA Triad," 30 June 2008. [Online]. Available: www.techrepublic.com/blog/it-security/the-cia-triad.

3 RTCA, Inc., "Airworthiness Security Process Specification," 2014, p. 21. [Online]. Available: https://standards.globalspec.com/std/9869201/RTCA%20DO-326.

4 Seventh Framework Programme, "Security Requirements of Vehicle Security Architecture," June 2011, p. 26. [Online]. Available: https://www.preserve-project.eu/sites/preserve-project.eu/files/PRESERVE-D1.1-Security%20Requirements%20of%20Vehicle%20Security%20Architecture.pdf.

5 Department of Defense, "DoD Information Security Program: Protection of Classified Information," p. 105. [Online]. Available: https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodm/520001_vol3.pdf.

6 National Institute of Standards and Technology, "Guide to Industrial Control Systems (ICS) Security, Special Publication 800-82," June 2011, p. 1. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf.

7 "Content of Premarket Submissions for Management of Cybersecurity in Medical Devices." [Online]. Available: https://www.fda.gov/media/119933/download.

8 Department of Health & Human Services, "Guidance to Render Unsecured Protected Health Information Unusable, Unreadable, or Indecipherable to Unauthorized Individuals," [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/breach-notification/guidance/index.html.

9 F5 Labs, "What Is the CIA Triad?" https://www.f5.com/labs/articles/education/what-is-the-cia-triad.

10 National Security Agency, "Defense in Depth," [Online]. Available: https://apps.nsa.gov/iaarchive/library/ia-guidance/archive/defense-in-depth.cfm.

11 Security Snobs, "Types of Locks," [Online]. Available: https://securitysnobs.com/Types-Of-Locks.html.

12 B. Schneier, "Class of Algorithms," in *Applied Cryptography*, John Wiley & Sons, Inc., 1996, p. 217.

13 E. Barker, "Recommedation for Key Management—Part1: General (Revision 4)," September 2015, p. 62–66. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

14 E. Barker, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths," November 2015, p. 7, 9-10. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf.

15 Barker, p. 33–46.

16 National Institute of Standards and Technology, "Federal Information Processing Standards Publication 197," 26 November 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

17 Barker, p. 7.

18 "INFORMATION SECURITY: FAA Needs to Address Weaknesses in Air Traffic Control Systems," Government Accounting Office, 29 January 2015. [Online]. Available: www.gao.gov/products/GAO-15-221.

19 Trusted Computing Group, "Secure Embedded Platforms with Trusted Computing: Automotive and Other Systems in the Internet of Things Must Be Protected," June 2012. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/Secure-Embedded-Platforms-with-Trusted-Computing-Automotive-and-Other-Systems-in-the-Internet-of-Things-Must-Be-Protected.pdf.

20 Committee on National Security Systems, "Use of Public Standards for Secure Information Sharing," 1 October 2012. [Online]. Available: http://www.cnss.gov/CNSS/issuances/Policies.cfm.

21 National Security Agency, "Guide to Industrial Control Systems (ICS) Security," May 2013. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf.

22 M. Scholl and A. Regenscheid, "Safeguarding Data Using Encryption," 2014. [Online]. Available: http://csrc.nist.gov/news_events/hipaa-2014/presentations_day1/scholl_hipaa_2014_day1.pdf.

23 National Institute of Standards and Technology 2015, p. 3.

24 Internet Engineering Task Force, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," February 2011. [Online]. Available: https://tools.ietf.org/html/rfc6071.

25 Network Working Group, "The Transport Layer Security (TLS) Protocol, Version 1.3," August 2018. [Online]. Available: https://tools.ietf.org/html/rfc8446.

26 Network Working Group, "HTTP Over TLS," May 2000. [Online]. Available: https://tools.ietf.org/html/rfc2818.

27 K. McKay and D. Cooper, "Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations," August 2019. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf.

28 P. R. Egli, "Internet Security," 15 February 2011, p. 39. [Online]. Available: www.indigoo.com/dox/itdp/10_Security/Internet-Security.pdf.

29 United States Computer Emergency Readiness Team, "Understanding Denial-of-Service Attacks," 20 November 2019. [Online]. Available: https://www.us-cert.gov/ncas/tips/ST04-015.

30 T. Hardjono and G. Kazmierczak, "Overview of the TPM Key Management Standard," Trusted Computer Group, p. 6. [Online]. Available: https://trustedcomputinggroup.org/resource/overview-of-the-tpm-key-management-standard/.

31 T2080 Product Brief, April 2014. [Online]. Available: https://www.nxp.com/docs/en/product-brief/T2080PB.pdf.

32 R. Kissel, A. Regenscheid, M. Scholl, and K. Stine, "Guidelines for Media Sanitization," December 2014. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-88r1.pdf.

33 SANS Institute, "Covert Data Storage Channel Using IP Packet Headers," Feb 2008. [Online]. Available: https://www.sans.org/reading-room/whitepapers/covert/paper/2093.

34 J. Thyer, "Covert Timing Channels Design and Detection," [Online]. Available: https://www.sansorg/reading-room/whitepapers/covert covert-data-storage-channel-ip-packet-headers-2093.

35 A. Mileva, A. Velinov, and D. Stojanov, "New Covert Channels in Internet of Things," Securware2018, [Online]. Available: http://eprints.ugd.edu.mk/20423/1/securware_2018_2_10_30122.pdf.

36 E. Barker, M. Smid, D. Branstad, and S. Chokhani, "A Framework for Designing Cryptographic Key Management Systems," August 2013. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-130.pdf.

37 J. M., "Intel Digital Random Number Generator (DRNG) Software Implementation Guide," Intel, 15 May 2014. [Online]. Available: https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide.

38 Freescale Semiconductor, "QorIQ P4080 Communications Processor Product Brief," September 2008, p. 23. [Online]. Available: http://cache.freescale.com/files/32bit/doc/prod_brief/P4080PB.pdf.

39 J. M., "How to use the RDRAND engine in OpenSSL for random number generation," 30 July 2014. [Online]. Available: https://software.intel.com/en-us/articles/how-to-use-the-rdrand-engine-in-openssl-for-random-number-generation.

40 Information Technology Laboratory, "Federal Information Processing Standards Publication, Secure Hash Standard (SHS)," August 2015. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

41 J. Boyens et al, "Supply Chain Risk Management Practices for Federal Information Systems and Organizations," April 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-161.pdf.

42 G. Coker et al, "Principles of Remote Attestation," [Online]. Available: http://web.cs.wpi.edu/~guttman/pubs/good_attest.pdf.

43 Network Working Group, "The Kerberos Network Authentication Service (V5)," July 2005. [Online]. Available: https://www.ietf.org/rfc/rfc4120.txt.

44 "Firewall (computing)," [Online]. Available: https://en.wikipedia.org/wiki/Firewall_(computing).

45 McAfee Labs, "141 Cybersecurity Predictions for 2020" [Online]. Available: https://www.forbes.com/sites/gilpress/2019/12/03/141-cybersecurity-predictions-for-2020/#623f8cb61bc5.

46 "Tamperproofing" [Online]. Available: https://en.wikipedia.org/wiki/Tamperproofing.

47 National Electrical Manufacturers Association, "Manufacturer Disclosure Statement for Medical Device Security," 2019. https://www.nema.org/Standards/Pages/Manufacturer-Disclosure-Statement-for-Medical-Device-Security.aspx.

48 K. Herold, "Choosing Linux for Medical Devices," [Online]. Available: www.windriver.com/whitepapers/choosing-linux-for-medical-devices/White_Paper_Choosing_Linux_for_Medical_Devices.pdf.

WIND

Wind River is a global leader in delivering software for the intelligent edge. Its comprehensive portfolio is supported by world-class professional services and support and a broad partner ecosystem. Wind River is accelerating digital transformation of critical infrastructure systems that demand the highest levels of safety, security, and reliability.