SECURITY TENETS FOR
LIFE CRITICAL EMBEDDED SYSTEMS

November 20, 2015

TABLE OF CONTENTS

A. EXECUTIVE SUMMARY

Critical embedded systems are life-critical or safety-critical systems whose failure or malfunction may result in:

- Death or serious injury to people,
- Loss or severe damage to expensive equipment,
- Environmental harm, or
- Large, non-recoverable financial losses.

Additionally, the definition is even further restricted to high-performance, distributed computing systems that:

- Manage high-bandwidth I/O communications,
- Involve real-time processing, and
- Are environmentally constrained in Space, Weight, and Power (SWAP) consumption/dissipation.

The focus of this paper is on those critical embedded systems that are life critical.

The importance of designing security into life critical embedded systems from the beginning is increasingly evident as more devices are becoming interconnected as we move closer to an Internet of Things (IoT). As we apply smart, connected, embedded computing devices to improve systems with life critical roles, obviously this needs to be done responsibly. These devices have the potential to better mankind, but also the potential to be co-opted by malicious parties and do grave harm. Unfortunately, simple, clear, and current "security tenets" are not yet well articulated for building life critical systems with embedded computing capabilities. Much of the guidance that has been written now fails to address both the increasingly sophisticated threats which these systems face, requiring security to be embedded more deeply in the system. The current and future generations of embedded computing technology will continue to cut across industries "horizontally", bringing to light once again the need for greater security and safeguards in these devices. In that context, this document suggests basic security tenets to ensure that all life critical embedded systems across all industries have a common understanding of what is needed to protect human life, where it depends on or can be endangered by embedded computing.

This document should not be taken as regulatory in any sense. Each industry will need to evolve to conform to these tenets. However, the timelines and details of such evolution are to be determined elsewhere, not in this document. This document simply defines a safer end-state, not the route for each industry to get there.

B. INTRODUCTION

The Internet of Things (IoT) will add 50 billion devices to the Internet by 2020 [1]. Smart devices are already playing life critical roles in vastly different areas ranging from traditional Supervisory Control and Data Acquisition (SCADA) to modern Industrial Control Systems (ICS), connected cars, and countless areas of medicine, such as patient monitoring and embedded medical devices. These facts are driving the necessity of properly embedding security in the foundation of these devices. This is a difficult task with two primary hurdles. First, many existing devices are unable to fully support security. Replacing or upgrading these will be costly and slow. Second, whatever changes are made must allow device vendors to keep pace with rapid advancements in the technology and attack spaces; few vendors choose to be security leaders. The security of these devices will increasingly have national security implications as well, given the scale at which life critical technologies are being deployed. It is important to note that although this paper refers specifically to devices, and not products, a life critical device could be part of a larger ecosystem of devices that could negatively affect the security of the life critical device itself.

As with all systems, the security and safety of these systems are subject to the "weakest link" challenge. Thus, an additional focus on overall system integrity and how individual components and subsystems interact is key to avoiding situations where "the sum of the parts is a *hole*." Today, security is inadequate in many of these smart and embedded devices. The establishment of a set of core security tenets that manufacturers should incorporate into their devices is needed. These defining principles, or tenets, help establish best practices for ensuring that human life, information, and infrastructure will remain safe and secure. This paper is the result of our efforts to identify and explain these tenets.

The security tenets described in this paper were chosen to help "raise the bar" for security in the life critical embedded systems space, but they can do much more. There are currently a fair number of best practices and standards available for a wide range of industrial and consumer spaces. However, there are few that have been identified as common across industries. Much of the guidance available today was developed when security was viewed from a holistic system perspective, rather than building the security into the individual components. Since most threat models are soon overtaken by technological advances, market pressures, and societal changes, this paper focuses instead on those guiding principles which can improve the security of life critical embedded systems (and potentially many other industries).

We acknowledge that many deployed life critical embedded systems will not meet all of the tenets specified in this paper. Where possible, those systems should be upgraded to comply with this guidance as quickly as possible. By their very nature, life critical embedded systems are often exposed to new threats due to their use as control points and typical high interconnectivity. Where such systems cannot be upgraded to comply with all the tenets, the systems should be phased out and replaced on a priority proportionate to the magnitude of their impact.

The word "evolve" was purposely chosen in this document because in many situations a dash towards improved security or safety could favor one of these goals at the expense of the other. Specifically, the reader is cautioned against assuming that security and safety are equivalent concepts. While they are often related and combine to provide the appropriate degree of each, they have different motivations. One can imagine a system that is so secure that its operating parameters could not be changed without a specific key or password, even in an emergency. This could quickly lead to a severe threat to the safety of the operators, customers, equipment, or the environment. Similarly, the overzealous pursuit of safety could result in a system that was neither secure nor efficiently operable. When considering the replacement of life critical embedded systems to improve safety and security, the goal should be to achieve a harmony between them that is appropriate for the environment.

The guidance in this document is framed to shape certifications and specifications to come. The strength of word choice (e.g., MUST or SHOULD) indicates the criticality and, thus, the priority, of implementing a particular tenet. Use cases are included in Appendix A to illustrate the potential consequences of not implementing the tenets.

The tenets are organized into seven areas:

- General Security
- Communications Security
- Boot-Time Security
- Run-Time Security
- Managing Life Critical Embedded Systems Securely
- Security for Back-end Systems
- Monitoring for Advanced Threats

The tenets emphasize system integrity for a few reasons. First, strong guidance already exists in many communities for engineering resilient, high-availability fault-tolerant systems in the face of natural and man-made risks. Second, the façade of availability presented by systems and components whose integrity is compromised can often be more lethal than situations in which failure of those components and systems is quickly recognized. Third, strong guidance exists for ensuring confidentiality of information, but not all life critical embedded systems depend on confidentially of information. In fact, confidentiality and privacy are occasionally sacrificed to ensure integrity and availability of life critical embedded systems. Throughout this document, the word "compromised" is synonymous with "corrupted or destroyed" and is considered an unacceptable outcome. Consistently, where system integrity of life critical embedded systems is compromised, the potential for loss of human life, loss or severe damage to equipment or environmental harm is greatly increased.

C. TENETS

1. GENERAL SECURITY

a. Systems MUST have documented threat models.

The imperatives in this section of general security tenets are cornerstone starting points. Implementation decisions should depend on a detailed threat model as well as the typical concerns of energy constraints and processing capabilities. Ideally, these dependencies should be prioritized over cost wherever possible. Good guidance on formal threat modeling can be found with a quick web search.

While threat models are always going to be part of a larger ecosystem, focusing on protection against those threats should be addressed. Threat models should capture all assumptions and consider all aspects of the system. For example, the models should include supply chain complexities where some equipment or components are often supported by third parties who might be trustworthy or untrustworthy to varying degrees. While the threats to supply chains and other threats, such as malicious insiders, are beyond the scope of this work, they too should be assessed and included in a threat model for life critical embedded systems. Such an assessment could lead to changes in procurement policies, personnel training, authentication protocols, and access control management.

The threat posed by physical access to a life critical system is based on the specific environment and how the system is used and maintained. Protection of systems against physical tampering is a difficult undertaking, potentially resulting in much higher system purchase prices and operational costs. Concerns related to physical tampering are best approached through policy controls. For example, physical access to process control systems in a refinery must be well-defined and enforced.

It is no longer sufficient to consider any life critical embedded system in an enclave as adequately isolated from the rest of the world. Air gapped solutions have been proven to provide a false sense of security [2] [3], so it is now necessary to assume threats will penetrate the enclave and insiders may accidentally or purposefully go around the air gap. Air gaps should be viewed as part of a defense-in-depth solution, rather than a single solution that would prevent or deter an adversary from gaining access to a system or network. Security must be engineered to protect "from the inside out" to provide additional security layered on the traditional "outside in" security engineering.

As life critical embedded systems become more capable, it becomes increasingly important to consider each system end-to-end. For instance, in some cars a Tire Pressure Indicator (TPI) originally only informed the driver of a need to change a tire. Now that same TPI may send its data directly into the digitally controlled braking systems. For each actuator (the control unit that makes changes to the system based on instructions from a controller), one must consider the full waterfall of sensors and analysis that contribute to each decision. These components no longer exist in isolation.

Furthermore, back-end systems can affect the threat model of the system as well. It's always possible that the back-end systems may go offline or, as with mobile systems, may be out of communications range for substantial periods of time. The threat model should include the entire lifecycle of the device and the subsequent devices which it depends upon for operation. The threat model should address what happens if the back-end system is retired permanently or its sponsoring organization is unable to maintain it due to bankruptcy or other conditions rather than capturing these situations under "fail safe" behavior.

It is also important to recognize and model the reality that in many life critical embedded systems some components are far more life critical that others. For instance, in an unattended vehicle, the emergency brake is more life critical than the air conditioning (AC) system. In safety engineering, all things electrical, even traditional AC systems carry specific fire risks, particularly in the event of a crash. When planning for security, seemingly benign things like streaming connections to the vehicle's radio, as well as the remote (cellular) ability to start the AC system, may represent attack vectors for the other systems in the car and, as such, must be properly modeled and any resulting security risks properly mitigated.

Ideally, a proper threat model will help induce a policy of separation between critical and non-critical systems. This concept is sometimes referred to as red/black separation, where signals and systems that carry sensitive information and control safety critical systems are kept physically separate from non-sensitive systems. As a threat model is developed, the sensitive components of a system should be identified and ways to keep these components physically, or to a lesser extent, at least logically separated from less sensitive components should be developed and implemented. As an example, one design option would be to have a car's entertainment system, which may be connected to the Internet (e.g., for receiving streaming media content), kept completely separate from the car's drive-by-wire controllers. However, while maintaining a strict policy of separation is ideal, there may be a need for the interconnection of systems to enhance safety and features. When such systems are connected, extra precautions should be taken to ensure logical separation of sensitive and non-sensitive components.

Threat models must recognize that some systems will need to be in place for decades, while others may refresh annually or more frequently. The imperatives for an update mechanism help mitigate some risks, but they do not address the vulnerabilities introduced when non-updatable legacy systems are connected directly to modern systems. Life critical embedded systems should be engineered to include enough compute capacity for stronger cryptographic and run-time protections that will need to be added within the lifetime of the systems. Ideally, life critical embedded systems would include a hardware root of trust and system integrity, as without such system hardening, updates could be unreliable or untrustworthy. Even with these security mechanisms, systems may be compromised or simply fail. Not

addressing remediation and failure plans can endanger lives or incur exorbitant, avoidable costs associated with replacing the system when threats get ahead of the deployed hardware. The resulting threat models can be used to instill remediation plans inclusive of the update cycles and process flow.

b. Systems MUST be engineered to fail safely.

This security guidance is in addition to and not in place of traditional safety engineering. Traditional safety engineering recognizes that distributed systems and their failure modalities can be complex. Systems need to be engineered to fail gracefully, and important decisions like "fail open versus fail closed" need to be made carefully. Systems need to be engineered to "do no harm" even when things are going wrong quickly. Simple primitives can be tremendously invaluable, including a fully automatic (safe) shutdown procedure that is easily initiated from any of many emergency stop buttons throughout a facility. As the complexity of systems and requirements continue to increase, fast, simple, and safe shutdowns become absolutely crucial, regardless of whether they are triggered by a manual stop button or automated detection of unstable states. Complexity is just one of many reasons why security and safety within systems and their individual components must be considered and decided in the design phase as many aspects cannot be "bolted on" later. Specific devices may have standards that apply to them, and there may be more detailed guidance. For example, for the industrial control space see the NIST Special Publication 800-82 series.

c. The data usage, safety, and privacy aspects of life critical embedded systems MUST be clearly documented in plain language.

Ecosystems that employ life critical embedded systems must clearly articulate the security and privacy risks in a well-organized manner using simple terms. It is expected that life critical embedded systems must also articulate to the builders and integrators of systems and shared environments, the security and privacy threat models and risks. This ultimately allows for users and owners to make a clear, informed choice in participation. Many people come near life critical embedded systems, regardless of whether those life critical embedded systems are embedded in a car, or an airplane, or a factory floor. In each case, these systems are now making complex decisions. People must know what to expect of such critical systems. For example, a vehicle's Wi-Fi system may automatically connect to

open wireless systems in order to send information outbound or request information or updates. This awareness includes clarity on life critical failure modalities of the system, as well as clarity on (otherwise) hidden dependencies such as the waterfall of sensors and analysis that contribute to each actuation (In case such a person was to note a sensor, processor, or actuator as faulty).

d. Devices MUST only run hardened code.

Before any code is signed for execution, it must be appropriately hardened through recognized industry best practices for manual and automated discovery of bugs and vulnerabilities, as well as remediation of the code. For the purposes of this paper, hardening is defined as securing code by limiting its attack surface. Additional remediation through obfuscation is desirable to slow reverse engineering but is not required. Compiler based techniques for hardening code is strongly desirable, among a variety of techniques for providing run-time protection of the system. While these tenets are not regulatory, an organization must establish a secure review process that would be engaged when major code changes are made throughout the lifecycle of a device.

e. Devices MUST enforce least privilege.

The concept of least privilege is that all system users and software operate with the lowest set of privileges needed to perform their duties. Further, access permissions are only available for the minimum amount of time needed. As the quantity and level of privileges increase, the attack surface and breadth of destruction increases. Employing least privilege provides many security benefits including limiting the impact of malicious or unwitting insiders. For example, consider the case of software that needs to access an area of memory. If the minimum set of privileges (e.g., read, write, execute) needed by the software when accessing the memory are read and write, the memory should be configured with only those two privileges. By not configuring the memory with the execution privilege, any rogue code written to memory cannot be executed.

Least privilege must be architected into the device or system being developed. For both major and minor components, it is important to identify the functions to be performed and the privileges needed for the functions to

operate. Also, consider the privileges needed for communications across components. When communications are necessary with devices or systems, take into account the level of privileges they use and, where possible, incorporate security techniques to mitigate any escalated privileges.

2. COMMUNICATIONS SECURITY

a. All interactions between devices MUST be mutually authenticated.

Authentication is the process of confirming the identity of an entity, such as a person, device, or data. Authentication of data refers to confirming the source of the data or validating that the data integrity has not been compromised. All data, commands, and requests must be mutually authenticated to be trusted. Any data, commands, and requests that cannot be authenticated should be ignored. Authentication of data, commands, and protocols matter because it is dangerous to accept data from unverified devices and/or services. Such data can not only corrupt or compromise devices, but also be the initial seed to grander threats and attacks. In addition to the authentication of data, it is also important to authenticate the devices, services, and systems that want to communicate, share data, and enforce control. Using strong mutual authentication to restrict such connections or communications at any layer helps protect the devices, services, and overall systems from such threats.

Two common ways to perform mutual authentication as part of the communications protocol are the use of secure sessions at the network link layer (e.g., IEEE 802.11i (for Wi-Fi), DTLS in Constrained Environments (DICE)) or via digital signatures on data, commands, and requests at the appropriate application layer.

A generally accepted digital authentication approach is based on elliptic curve cryptography (ECC), but over time other approaches may evolve. For additional information please see FIPS PUB 186-4: Digital Signature Standard (DSS) [4].

Note that from a performance perspective, mutual authentication is now feasible in extremely constrained devices where such authentication was previously infeasible. For example, recent implementations of the Elliptic

Curve Digital Signature Algorithm (ECDSA) have demonstrated that a number of 8-bit MCUs running at 8 MHz with only 32 kb of RAM are now capable of doing signature verification in a few seconds [5].

In addition to simple cryptographic authentication, it is desirable for devices to provide an attestation of their current security state. Depending on the threat model, this might actually be required. Such attestation could include digital fingerprints of the device's configuration and all code loaded, among other important security metrics.

In this tenet, authentication implies authorization. However, for clarity, connections and data are authenticated as coming from a given source. Once authenticated, the device must choose to trust or not trust that source based on not only authentication and attestation information, but also policy that should be updated over time. Such dynamic control of authorization and access control is crucial to safely handling components and devices that become compromised as part of a much larger system. Some means of efficiently providing such dynamic control include using mechanisms such as Online Certificate Status Protocol (OCSP) stapling, Trusted Network Connect (TNC), or other forms of dynamic Network Access Control (NAC) enforced either at the endpoint devices or at gateways between such devices.

b. Continuous authentication SHOULD be used when feasible and appropriate.

All data, commands, and requests should be continuously authenticated where feasible and appropriate. Authentication could be verified either at set intervals or with each set of communications processed as part of the communications exchange. Note that there could be an impact to performance depending on the functional requirements. Nonetheless, function and risk should be weighed as part of the feasibility and appropriateness of this tenet in light of the danger to human life.

c. All communications between devices SHOULD be encrypted.

The goal of encryption is confidentiality, while other cryptographic techniques are employed to provide authentication or fraud resistance/detection. Encryption protects the data so that only those who

have the appropriate keys may decrypt the data for reading or modification. This provides protection from eavesdroppers along the path between devices and/or systems. Such eavesdroppers might be able to maliciously leverage the data in some way. For example, captured process control information might provide hints to how some lucrative or dangerous process is accomplished, and perhaps how to interrupt its operation.

Note that not all devices and environments are immediately amenable to encryption, particularly in long life, low Central Processing Unit (CPU) power embedded systems. For those cases, a threat assessment is necessary to determine whether it would be prudent to replace/upgrade the device ahead of schedule or to introduce additional devices that can provide encryption capabilities for that device.

Encryption alone does not provide sufficient security. Encryption should be part of a comprehensive approach to raise the overall security posture of a system through improved confidentiality, authentication, and resistance to/detection of fraud, both on the local system as well as across a distributed computing environment. Using encryption in some parts of a system cannot make up for security and safety failures elsewhere in the system design.

3. BOOT-TIME SECURITY

a. Devices MUST NEVER trust unauthenticated data or code during boot-time.

Devices must never trust unsigned (i.e., unauthenticated) configuration files or any other form of unsigned data. To ensure confidence in the code's authentication (and the device's overall secure operation), devices must be designed to boot into a known good state.

Configuration files can be trusted if they are signed by an appropriate authority. They can be signed as part of a monolithic boot image or signed individually with appropriate protections against threats, including but not limited to rollback and replay and any other threats produced by diligent and professional security threat modeling (See Tenet 1a). Trusting an unsigned configuration file can result in malicious misconfiguration of the system, leading to any number of significant consequences.

A generally accepted authentication approach is the use of digital signatures based on ECC, but we recognize that over time other approaches may evolve. When verifying the signatures, the device would use a root of trust (e.g., programmed into Read Only Memory (ROM) or fusible bits) that must be under the control of the owner of the life critical embedded system. Allowing execution of unauthenticated code easily gives control of a device to aggressors. Depending on the threat model facing the system, the owner might choose to authorize all of the manufacturer's code to run on a given system or choose to put in place additional controls whereby the owner is able to control which code from the manufacturer is able to run on the device. All code must be authenticated and authorized before it is loaded for execution. This is true for the case of monolithic systems where the signature on the boot image includes signing the application on the device, as well as any operating system, firmware, and/or libraries. This is also true for systems where an application is signed separately from an operating system.

It is recognized that there may be challenges associated with implementing this tenet. For instance, there may be substantial additional engineering efforts needed to ensure secure boot of any microprocessor or MCU. However, secure boot and the imperative that devices must never be permitted to run unauthorized code are essential for life critical embedded systems to protect human life, equipment, and the environment.

b. Devices MUST NEVER be permitted to run unauthorized code.
Authorization is the process of granting or denying an entity, such as a person or process, access to a resource or the ability to perform an activity. Authorization is based on whether the person or process has the correct set of permissions or privileges needed.

This tenet assumes Tenet 3a is being implemented correctly. Devices must never run anything other than authenticated code, authorized by the party responsible for managing the life critical embedded system. Typically, this party is simply referred to as the owner of the life critical embedded system, regardless of any financial ownership and/or property rights. Code refers to both firmware and software.

4. RUN-TIME SECURITY

a. Devices MUST mitigate run-time security risks, including malicious data. Unfortunately, even after devices are booted into an authorized configuration, and even if the code has been reviewed and hardened via manual and automated best practices, the code can still have unknown runtime vulnerabilities that must be mitigated. Mitigation can include policy-based lockdown of resources such as processes, or content based filtering of potentially dangerous data. This mitigation can be done via techniques, such as including some form of an intrusion prevention system (IPS) in the device's network stack or ensuring that the device is only capable of connecting to a gateway that provides such an IPS function. Other techniques include advanced methods for using memory introspection to ensure that executable code changes remain unchanged from boot. Additional techniques include host-based behavioral methods, application sandboxing, application whitelisting, device and configuration control, reputation based techniques, and cryptographic protections on run-time (not just boot-time) resources. Through one mechanism or another, run-time security of devices in a life critical embedded system should be continuously monitored in a secure manner and continuously verified. Specific mechanisms for providing run-time security will vary widely by system architecture and environment.

There may be times when a suspected malicious access attempt is blocked, yet the attempt was both safe and legitimate. In this context, extreme care must be taken in protecting any life critical availability requirements while attempting to mitigate run-time risks. In extreme cases, it can be acceptable to build in a mechanism capable of blocking such access, but configured to only monitor such access until risk levels change.

Denial of service attacks may also be mounted against life critical embedded systems. For example, an adversary may attempt to saturate (i.e., flood) a target device with communications requests to cause it to be unable to respond, or perhaps drain a target device's battery (i.e., a battery exhaustion attack). Protections should be in place to mitigate these sorts of attacks. Any solution must let the legitimate traffic flow while blocking the malicious attack traffic.

It is recognized that industry's ability to protect activities at run-time is currently limited. Best efforts must be taken to address risks as best as possible. However, some threats will still succeed, and for that reason additional monitoring and mitigation is required for advanced threats as described in Tenet 7a.

b. Devices SHOULD NEVER trust unauthenticated data during run-time.
In distributed systems, devices often receive data from other devices. Consistent with the imperative that all interactions between devices MUST be mutually authenticated, devices must never trust unsigned data. In this context, as a minimum, each device must confirm the pedigree of data coming from any device. Additionally, it is preferable that, where possible, the pedigree flows with the data from the original sensor collection and through any handler devices, gateways, translation, and subsequent processing. Each device handling the data appends its signature for any transformations and includes the original data when possible. This strategy best mitigates the risk of the data being tampered in transit, as well as at rest and/or in processing by a compromised device.

It is recognized that this strategy is rarely feasible in energy constrained systems that depend entirely on batteries or energy harvesting. In the case of legacy systems, it is expected that they will be upgraded overtime to meet this tenet.

c. When used, cryptographic keys MUST be protected.
Protection technologies will vary based on the threat model and system architecture, but cryptographic keys used for authentication must be protected from leakage. Please note that while it is important to protect private (secret) keys from leakage, it is equally imperative that public (authorized) keys must be protected from tampering, particularly for keys (or certificates) used as roots of trust in verification of other parties' certificates or used in verification of signatures on signed code. It should not be possible for an adversary to swap roots of trust or append their root of trust to any device's truststore.

Hardware protection for keys is desirable and might be required depending

on the threat model. Specific protection technologies include but are not limited to Trusted Platform Modules (TPM), various types of security architectures, and physical countermeasures to side-channel analysis and both non-destructive and destructive types of reverse engineering.

5. MANAGING LIFE CRITICAL EMBEDDED SYSTEMS SECURELY

a. Devices and systems MUST be built to include mechanisms for in-field update.

Vulnerabilities will be found in these devices, and they will need to be patched to stay safe and secure. Additionally, many of the run-time protections previously mentioned often require updates to security content. All such updates must be done securely.

Over time, aggressors will reverse engineer devices, discover vulnerabilities, and exploit those vulnerabilities. For these reasons, all devices must include the ability to be quickly updated whenever vulnerabilities and/or exploitation are discovered.

It is recognized that such updates are difficult and energy consuming in energy limited devices that are either battery constrained or constrained by energy harvesting. It is also recognized that such battery constrained devices often need small, specialized batteries to last years or decades. In such contexts, changing an entire firmware image could drain months or years of battery life or, in worst cases, if performed poorly, over half the battery life. Many aspects of the embedded world of IoT are often radically different from the simpler world of traditional Information Technology (IT).

The ability to update these devices is essential to ensuring the continued proper and secure operation of these devices over the long term. Further, these update mechanisms must be built into each device from the beginning since adding them to existing systems would most likely be less effective, less reliable, and less secure, if even possible. For such highly constrained devices, it becomes crucial to include some form of update management process that ensures updates proceed smoothly and that partial, failed, or rolled back updates do not endanger the device's functionality or place the device into a vulnerable or dangerous state.

In-field updates are one component of an overall lifecycle management plan. In cases where in-field updates are not possible, alternative practices for ensuring the continued security and safety of those devices must be in place. For these systems, an accelerated replacement schedule should be established– essentially associating an "expiration date" with such systems. Short-term extensions to this deadline should be provided if no suitable replacements with improved life critical capabilities are available at that time.

b. Devices and systems for managing updates MUST be mutually authenticated and secured.

As embedded systems and devices are deployed in remote and not easily accessible locations, it is required that the software updates (whether from a general feature update or due to a security patch) be done using remote communications. While it is understood that the system infrastructure will be aware of the deployed devices it manages, the devices themselves must also have a mechanism to acknowledge and authorize the infrastructure communicating with it, especially as its configuration, software, and firmware can be affected. Without the means for the device to authenticate and authorize the system, the device can be vulnerable to anyone or any system configuring and running any software on the device. Visibility into a device's identity is critical to the life cycle management of the device.

Devices and systems should avoid communications with legacy and non-updatable devices and systems. Communication with devices that are unknown, have little to no security, or cannot be updated should rely on the ecosystem to establish trust, relationships, and verification of communications. Devices should avoid accepting data from other devices with unknown security properties.

6. SECURITY FOR BACK-END SYSTEMS

a. Systems communicating with life critical embedded system devices MUST be protected in accordance with industry best practices.

Many IoT systems use cloud-based services and technologies. As IT and Operational Technology (OT) collide in both IoT and life critical embedded systems, it is important to remember that, where a device is driven by a server or cloud-based service, failing to protect that server/service can

produce outcomes equivalent to failing to protect the device. Fortunately, there are many best practice guidelines for protecting such back-end servers and cloud-based services. For example, the Open Web Application Security Project (OWASP) and SafeCODE provide valuable guidance in addition to vertical specific guidance. Some of these organizations are currently developing guidance for embedded systems. For instance, organizations like the Trusted Computing Group (TCG) have developed technologies to cryptographically attest the state of servers in the cloud. Trustworthiness assessment of cloud-based services through attestation should be part of best practices for protecting IoT devices.

7. MONITORING FOR ADVANCED THREATS

c. Systems MUST be monitored for threats capable of defeating or avoiding these tenets.

Unfortunately, even with all of the previously mentioned tenets taken into account, some of the most advanced threats, such as insiders, will still be capable of defeating any best practice. To mitigate the risks from such threats, it is important that life critical embedded systems include a monitoring system where device states and communications between devices can be monitored. Then, if an advanced threat is discovered, it can be dynamically tracked and potentially mitigated via remediation. Such a monitoring capability will require strong data collection and analytic capabilities akin to those of Security Operations Centers (SOC) and/or Computer Emergency Response Teams (CERT). It is also important to ensure that a mitigation plan is in place when an issue occurs.

The capability to monitor will also require intimate familiarity with the unique aspects of the life critical embedded system and the ability to investigate and act on timescales appropriate for the specific life critical embedded system being monitored. Such monitoring will need to span in-field devices and any servers and/or cloud-based services driving them.

Note that for systems already deployed, particularly those with devices that are extremely limited and not easy (or possible) to update, such monitoring can sometimes be achieved by deploying new devices to listen and/or sniff between already deployed devices without disrupting them.

The tenets included in this whitepaper provide a baseline level of security. They should not be viewed as all that needs to be done to secure life critical embedded systems. Every organization must follow industry best practices and hygiene recommendations to protect itself against complex threats such as Advanced Persistent Threats (APT), Distributed Denial of Service (DDoS) attacks, Cross-site Scripting, and Structured Query Language (SQL) Injection.

APPENDIX A - Use Cases

INTRODUCTION

The use cases that follow were designed to demonstrate real-world security threats to life critical embedded systems and generally to devices that are part of the Internet of Things (IoT). It is expected that these use cases will be disseminated, as they will have value and applicability in other contexts.

The use cases themselves are intended to be standalone scenarios that illustrate one or more poor security practices or common vulnerabilities that are often found in life critical embedded systems today. The use cases or "scenes" are tied together by an overarching story arc. The narrative is fictional, however, the ideas and concepts are grounded in actual incidents or demonstrated security hacks.

Throughout the narrative, each vulnerability is assigned a number which maps to one or more applicable security tenets. This is intended to show the value and subsequent need for implementing the security principles found in the paper. The mapping is listed in Appendix B.

USE CASE 1

The Widget Garage in the Bronx, New York is the main resource for many New York City (NYC) taxi's routine maintenance, service, and repair needs. The garage also services ambulances as needed. In July, a number of taxis and limousines made their way through this maintenance depot for common maintenance items like new brake pads, oil changes, general repairs, and any on-board computer system firmware and Technical Service Bulletin (TSB) updates. Each vehicle is typically triaged and sent through different bays of the maintenance departments for service. One bay in the garage usually performs the on-board computer system maintenance related to firmware and TSB updates. Throughout the months of July and August, a significant number of the taxis, limousines, and ambulances went through this bay for routine updates to their control systems and creature amenities.

One of the recently installed creature amenities included an in-vehicle Wi-Fi entertainment system for a more interactive rider experience. This Wi-Fi system operated in a mesh configuration for connectivity, load, and cost, but eventually

communicated back to strategically placed base stations to provide rider internet connectivity and dispatch communications. This mesh environment also enabled car-to-car communications to indicate the speed and flow of traffic amongst each of the taxis that communicate back to a number of the base stations that then communicate back to dispatch. Some of these systems slightly adjusted the acceleration available to each vehicle to allow for more fuel/battery efficiency. [1] The Wi-Fi systems in the vehicles integrated directly to the computer based system controls on the taxis and limousines in order to report accurate and detailed fuel usage and battery charging statistics back to dispatch and the garage. [2]

A terrorist cell consisting of an unknown number of industrial and consumer control system hackers has spent months planning an attack on the Lincoln Tunnel. Through their planning, they have researched and analyzed traffic flows and patterns through the tunnel at various times to determine the optimum time to strike. This cell, calling itself "Team F", has implanted one of its members as a mechanic at the Widget Garage. While employed at the garage for a few months, Team F's member has modified the code within the acceleration items and braking items used by the taxis. [3] They also modified the code for the limousines to allow remote execution of braking. [4] The limousines' braking firmware also had elements and updates that were shared with the ambulances from the manufacturer. [5]

This modified code allows for direct communications via the Wi-Fi connection utilized as part of the creature amenities in the vehicles. [6] This direct connection also allows for communications to the Controller Area Network (CAN) bus units in each vehicle. [7] Access to the CAN bus allows for direct control of acceleration and braking elements of the vehicles. [8] Furthermore, the mesh networking elements allow for communications from the CAN bus unit back to base stations and dispatch.[9]

This modified code was utilizing a revoked certificate that was previously valid, signed, and stolen from the CAN bus controls manufacturers earlier in the year. [10] The manufacturer would eventually realize that its signing certificates were stolen in the months after the attack, which will prompt it to issue a TSB which forces an update to the Certificate Revocation List (CRL).

The Team F implant placed a number of firmware update SD Cards in locations

around the garage with the latest dates and revisions for April/May mimicking the style used by the vehicles' manufacturers for look and appearance of the SD Cards. [11] Throughout the months of July and August, a large number of these vehicles were brought in for updates to their on-board computer systems, battery systems, braking systems and in-car Wi-Fi entertainment systems. There were no updates that failed, as the certificate seemed valid. [12] The majority of updates were performed utilizing the SD Cards containing the modified code.

USE CASE 2

At 4:00 p.m. on the Friday before the Labor Day holiday weekend, Team F positioned itself at the north end of the Lincoln Tunnel in a vehicle traveling back and forth through the tunnel. They had a specially configured PWNIEPRO device with customized packages and a Software Defined Radio integrated. Team F's objective was to create a significant vehicle accident inside the tunnel with an initial maximum casualty impact, followed by a disruption in traffic for those trying to leave the city for the holiday weekend.

They wait for a number of the serviced taxis and limousines that would be from the servicing company of Widget's. As their PWNIEPRO gathers and connects to the Wi-Fi systems within each car, they verify connectivity to the CAN bus unit to confirm compromise and continuous connectivity. [13] Team F waits for compromised taxis traveling at speed with a few large tractor trailers close behind them at speed. They spot an opportunity to create the most impact with four taxis and two limousines traveling at speed while dispersed throughout the three tubes.

Through their continuous connections to the Wi-Fi and CAN bus systems, Team F executed a full brake on two of the taxis and an accelerate command on the other two taxis. [14] They also executed full braking commands on both limousines. [15] This caused a multiple car pileup at various places within each of the three tubes, and several vehicle fires dispersed throughout. All traffic traversing the Lincoln Tunnel came to a complete halt as several points along the 1.5 mile tunnel were blockaded with wrecked vehicles. Team F watched and confirmed the destruction from a vehicle in front of the fray, continuing on unscathed to the next stage of the attack.

USE CASE 3

Emergency response vehicles were dispatched within seconds via the closest fire and emergency response location. A few other members of Team F were also present directly on the traffic control system network via both physical locations and remote means. [16]

Over the past three months while the firmware updates were being deployed to the taxis, limousines, and ambulances, Team F was physically pulling manhole covers while dressed in apparent traffic control systems repairmen garb. This was done in very low security and low risk locations that would most likely share infrastructure with the same systems that would be utilized by the emergency response vehicles. [17] Team F placed a few wireless routers on network equipment that is used for the traffic control systems, including traffic cameras, via these physical attacks. [18] Only three routers were needed to gain persistent connectivity to the traffic control systems.

The traffic light systems ride on a network that is not access control listed off from the video control systems. [19] This allowed Team F to easily manipulate the traffic light control system from both an emergency lighting standpoint as well as a maintenance mode standpoint, placing lights to blink in directions that are contrary to an emergency medical response.

Furthermore, the camera systems in and around the tunnel often utilize a set of video communications that is claimed to be obfuscated end-to-end. However, often times the methods of obfuscation are actually utilizing wrapper based end-to-end communications. These common headers are well known within the traffic control systems community. The fact that these are known headers allows for stripping of the wrappers on the communications packets and thereby collecting the raw video feeds in an un-obfuscated fashion. Consequently, this communications obfuscation is no replacement for end-to-end encryption. [20]

Team F has done this packet stripping and created a number of traffic video recordings that indicate normal activity, including some with emergency vehicles passing by. [21] They placed these recordings into the camera network's live stream for critical spots during the responder's route. This created confusion and

miscommunications between the dispatchers and the emergency responders.

Team F had another method of attack to others already impacted by the traffic system. This consisted of the mesh networks that the remaining compromised taxis and limousines used to communicate amongst one another which allowed for more direct control of the CAN bus units. [22] This mesh network allowed for Team F to randomly apply brakes and acceleration throughout any of the compromised vehicles and the connections they could acquire via their customized PWNIEPRO. [23]

The mesh network also allowed for communications back to dispatch on the vehicles that were not compromised through the firmware update affecting each vehicle in the fleet's mesh system. [24] These communications allowed for Team F to modify the run-time parameters reported back to the dispatch through the mesh systems and base stations. [25] The vehicles could erratically accelerate at different rates thereby creating yet more confusion and accidents throughout the routes to the two closest hospitals.

USE CASE 4

Intermingled with emergency responders were nearby NYPD police officers and transit authority officials on the north end of the tunnel. It was immediately clear to the local police that this was not an unlikely set of random accidents, but a coordinated terrorist attack affecting all three tubes simultaneously. Due to the nature of the incident and their recently updated standard operating procedures, the local authorities activated their toxic gas detecting wristbands before driving into the tunnel. The wristbands themselves wirelessly communicate with the patrol car's CAN bus system, sending clear text alerts automatically to dispatch for faster dissemination of chemical and biological detection. [26] Almost immediately after entering the tunnel, the sensors detect heavy concentrations of phosphine gas, a colorless toxic gas that is extremely flammable and explosive. The wristband worked flawlessly notifying the wearer and sending an alert to the local police station; the police then notify all local authorities and emergency personnel to not enter the tunnel without proper suits and respirators; significantly delaying any rescue attempts to injured motorists inside the tunnel.

What the authorities did not know is that there was no phosphine gas in the tunnel,

Team F successfully hacked the wristband and created a false positive which was then reported. Team F was able to accomplish this by using the PWNIEPRO to exploit the lack of access control on the wristband itself. [27] A quick sniff for open communications points in the area and interception of the wireless clear text communications between the wristband and the CAN bus system in the patrol car was all Team F needed to identify their next target. [28] Team F used this vulnerability to gain access to the wristband, root the device with the scripts on-board the PWNIEPRO, and generate a false positive alert which appeared to be authentic. [29] As accident victims were able to walk out of the tunnel with no visible signs of exposure, it took an additional 30 minutes before the confusion cleared and local authorities realized there was no toxic gas.

USE CASE 5

Many of the victims that could be removed from the tunnel were taken to the closest hospital via helicopter airlifts due to the traffic disruptions. This was a result that Team F anticipated and had smaller teams waiting at each location to execute the next set of events.

These smaller tactical teams had been running reconnaissance missions within the hospital to gather the types of medical devices they use, their network architecture mappings, and the most commonly used high-impact support devices as targets for a few months. [30] They decided initially to focus on the pumps used to deliver fluids, blood, and drugs to patients, heart monitors, and the medical record management system.

However, they had also decided to target any vulnerable machines they could find as a result of the tight integration with Bluetooth devices for dictation and wireless communications devices that would communicate with the crash carts and specialized pumps. [31] During their reconnaissance, they also noticed a number of HVAC systems, three of the five elevator systems, and emergency power systems sharing the same network. [32]

Team F has more targets time to execute on the targets.

APPENDIX B - Mapping of Use Cases to Tenets

1. 1.a, 1.b, 1.c
2. 6.a
3. 3.b, 7.a, 1.e
4. 3.b, 7.a, 1.e
5. 3.b, 7.a
6. 1.d, 3.a, 3.b, 4.b, 5.b
7. 6.a, 7.a
8. 1.b, 6.a, 7.a
9. 6.a
10. 4.c
11. 5.a, 5.b
12. 4.c
13. 2.a, 2.b
14. 4.a
15. 4.a
16. 1.a, 2.a, 2.c, 6.a, 7.a
17. 1.a, 7.a
18. 1.a, 2.a, 2.b, 2.c, 7.a
19. 1.a, 2.a, 2.b, 2.c, 7.a
20. 1.a, 2.c, 4.c
21. 1.a, 2.c, 4.c
22. 4.a, 4.b, 4.c
23. 1.b
24. 1.b, 4.a, 4.b, 4.c, 7.a
25. 2.a, 2.b, 2c, 4.a, 4.b, 4.c, 7.a
26. 2.c
27. 2.a, 3.a
28. 2.c
29. 4.a, 4.b
30. 6.a, 7.a

31. 1.a, 1.b, 2.a, 2.c
32. 6.a

APPENDIX C - Distilled Tenets

1. General Security
   a. **Systems MUST have documented threat models.**
   b. **Systems MUST be engineered to fail safely.**
   c. **The data usage, safety, and privacy aspects of life critical embedded systems MUST be clearly documented in plain language.**
   d. **Devices MUST only run hardened code.**
   e. **Devices MUST enforce least privilege.**

2. Communications Security
   a. **All interactions between devices MUST be mutually authenticated.**
   b. **Continuous authentication SHOULD be used when feasible and appropriate.**
   c. **All communications between devices SHOULD be encrypted.**

3. Boot-time Security
   a. **Devices MUST NEVER trust unauthenticated data and code during boot-time.**
   b. **Devices MUST NEVER be permitted to run unauthorized code.**

4. Run-time Security
   a. **Devices MUST mitigate run-time security risks, including malicious data.**
   b. **Devices SHOULD NEVER trust unauthenticated data during run-time.**
   c. **When used, cryptographic keys MUST be protected.**

5. Managing Life Critical Embedded Systems Securely
   a. **Devices and systems MUST be built to include mechanisms for in-field update.**
   b. **Devices and systems for managing updates MUST be mutually authenticated and secured.**

6. Security for Back-end Systems
   a. **Systems communicating with life critical embedded system devices MUST be protected in accordance with industry best practices.**

7. Monitoring for Advanced Threats
   a. **Systems MUST be monitored for threats capable of defeating or avoiding these tenets.**

APPENDIX D - References

[1] D. Evans, "The Internet of Things: How the Next Evolution of the Internet is Changing Everything," April 2011.

[2] K. Zetter, "Wired," 09 12 2011. [Online]. Available: www.wired.com/2011/12/worm-pentagon. [Accessed 18 11 2015].

[3] FireEye, "APT30 And The Mechanics of a Long-Running Cyber Espionage Operation," FireEye, Milpitas, 2015.

[4] National Institiute for Standards and Technology, "FIPS PUB 186-4: Digital Signature Standard (DSS)," 2013.

[5] K. MacKay, "micro-ECC: A small and fast ECDH and ECDSA implementation for 8-bit, 32-bit, and 64-bit processors," [Online].

[6] DHS, "Architectural Risk Analysis," 02 July 2013. [Online]. Available: https://buildsecurityin.us-cert.gov/articles/best-practices/architectural-risk-analysis/architectural-risk-analysis. [Accessed 2015].