VERVE

PROTECTING EMBEDDED SYSTEMS 101

From an asset owner's perspective:
Defining firmware and discovering embedded vulnerabilities to protect devices from exploitation

TABLE OF CONTENTS

Legal Notice....3

Embedded Vulnerabilities....4

- Introduction....4
- What is an embedded system?....5
- What is firmware?....8
- What is contained in firmware?....10
- How does firmware fit into a product How does it add security, functionality, or lack of?....12
- What kinds of vulnerabilities can be present in firmware of an embedded product?....13
- What are vendors doing, and where do firmware-related problems arise in the product?....14
- How do I determine if an embedded system has vulnerabilities?....16
- How do I protect embedded systems from vulnerabilities?....18

Author and Company Information....23

LEGAL NOTICE

All information products included in this document are provided “as is” for informational purposes only. Verve does not provide any warranties of any kind regarding the information contained herein, however, all reasonable efforts have been made to understand, collect, and aggregate publicly available information in this specific instance. All information is to be considered advisory, and all responsibility lays upon the reader regardless.

Further dissemination of this product is governed by the Traffic Light Protocol (TLP) marking in the footer of the document & any watermarks contained within. Suggested risk and impacts are indeed suggestions based on Verve expertise, available information, best-practices, and Verve cannot be held responsible nor make decisions ultimately on behalf of the reader or asset owner who owns all responsibilities.

This document is coded TLP-WHITE.

EMBEDDED VULNERABILITIES

INTRODUCTION

Awhile back I wrote about the fact that URG11 and network stack flaws are not anything new, and are miscreants left over from the 1990's and early 2000's – a period where these types of software flaws were rampant.

For the most part, these flaws represent an era that devices lacked proper robustness testing, and had customers obligated to trust the vendor's security practices. Whilst most of these were stranded in a land of security by obscurity or islanded ("air-gapped"), eventually were retired or rotated out of deployment and into the hands of researchers with ubiquitous network-stack protocol "fuzzers" (a strategy/application where you test all permutations of a protocol to see if there unintended effects or erroneous logic).

Yet, despite some of these vendors possibly having visibility or reports on these exact issues (or ones like them), stack-based vulnerabilities are commonly forgotten by vendor quality assurance and systems integration processes.

Well even if these systems are deployed in critical infrastructure, energy, oil & gas, manufacturing, building automation, or are consumer Internet of Things (IoT) products – the same issues are fundamentally present in all of those types of systems, and represent a variable level of opportunity & susceptibility to exploitation by a malicious entity.

Before I continue too far, I want to define the following questions:

- What is an embedded system?
- What is firmware?
- What does firmware contain? What are the components?
- How does firmware fit into a product - whether for function or security?
- How do I manage embedded system vulnerabilities?

This is a vocabulary problem that results from various disciplines: between software development, electrical engineering, systems management, vendors, market segments, and cyber security (IT, OT and even IoT/IIoT), but it is critical to understand the general concept on what is in an embedded system, regardless of the buzzword attached to it.

History Repeats

And as such, we arrived again at the same situation with RIPPLE20, so what gives? Why is it a gift that keeps on giving? And how can an asset owner keep tabs on embedded firmware vulnerabilities and reduce their OT cyber security risk?

Secondly, there are different kinds of vulnerabilities (or flaws), and these need to be discussed at some level because they are relationship to specific risk families, and/or vulnerabilities known or unknown that a device may encounter during its lifetime. Therefore, as an individual assessing risk, or as someone managing vulnerabilities, it is extremely important that you understand the concepts in this document so you can come to terms on the security realities for embedded devices.

Without further delay – let us talk "embedded systems 101" from an asset owner perspective.

WHAT IS AN EMBEDDED SYSTEM?

Without a historical lesson on embedded systems and how they have altered process automation – or even whether they are the basis for industrial process control and automation altogether, let's examine what an embedded system is, and what makes it different from that of a typical workstation or server.

At first glance, an embedded system appears to be like a typical computer – it often has a CPU, RAM, storage, and potentially network connectivity. However, the main general differences between a commodity system (e.g., a PC) is that:

A commodity system:

- **Usually running a commodity Operation System (OS) such as Windows or mainline Linux**
- **Has replaceable parts, and can be highly customizable** (e.g., an Administrator or Technician often can change policy configurations, install applications, or replace a hard drive)
- Has cyber security controls and technologies that are often enterprise or IT in origin
- Relatively easy to perform software upgrades
- Can be virtualized in many use cases
- Performs general computation and usage tasks fit for a wide audience

Embedded System

An embedded system is a collection of electronics and software that are packaged together into a specialized product for a specific purpose. It is often not commoditized and is highly proprietary.

An embedded system:

- **Usually running a specialized OS that has special customizations for specific hardware**
- **Features non-serviceable parts that are soldered onto a circuit board/module** or even built into a single chip (also called a System on a Chip (SOC)) - like a cell phone
- **Not typically virtualizable due to customized hardware** - software does not work well in every situation or scenario
- **Provides specific function at a specific frequency with minimal latency in Real-Time (RT)** (e.g., microseconds and not milliseconds)
- Less trivial to update software due to their deployment environments, but the complexity to build monolithic updates
- **Often designed to continuously monitor and interact with a cyber-physical solution through the usages of inputs and outputs (digital or analog)**

There are other differences and exceptions, but to many users and Administrators from the enterprise space, they may find it perplexing why one cannot rip out outdated equipment when there is an issue, or to even upgrade it. There are a number of reasons as to why (and some of those are specific to the OT realm), but I'll illustrate an example that may be familiar for those of you that have had Android cellphones (which is an embedded device) for quite some time, you may have noticed a few things:

- **Updates come in two formats usually** - the base OS and per application
- **The vendor of the phone often repackages or relies on other vendors as part of the product supply chain** (software and hardware) - this can complicate vulnerability management and the availability of fixes
- Previously, applications were "baked in" or contained into the base OS updates, which may or may not have ever been made available for an update, despite including cyber security updates
- New software might have adverse effects on "older" hardware, such as consuming additional resources and slowing down a "perfectly" good phone

- Upgrades can be tricky in some cases due to connectivity, per component contained within the device, update integrity and authentication - and even rolling back if there is a failure
- Ultimately, fragmentation of the ecosystem - the same SOC used in a phone packaged by several different vendors can have different versions floating around for the same setup

I realize a cell phone is not a Programmable Logic Controller (PLC) or "Smart" home thermostat, but I think this highlights a simplistic illustration of the challenges for embedded devices, but also the constraints around them. Looking down from a 65k' foot level, what is in a general embedded system in the following figure:

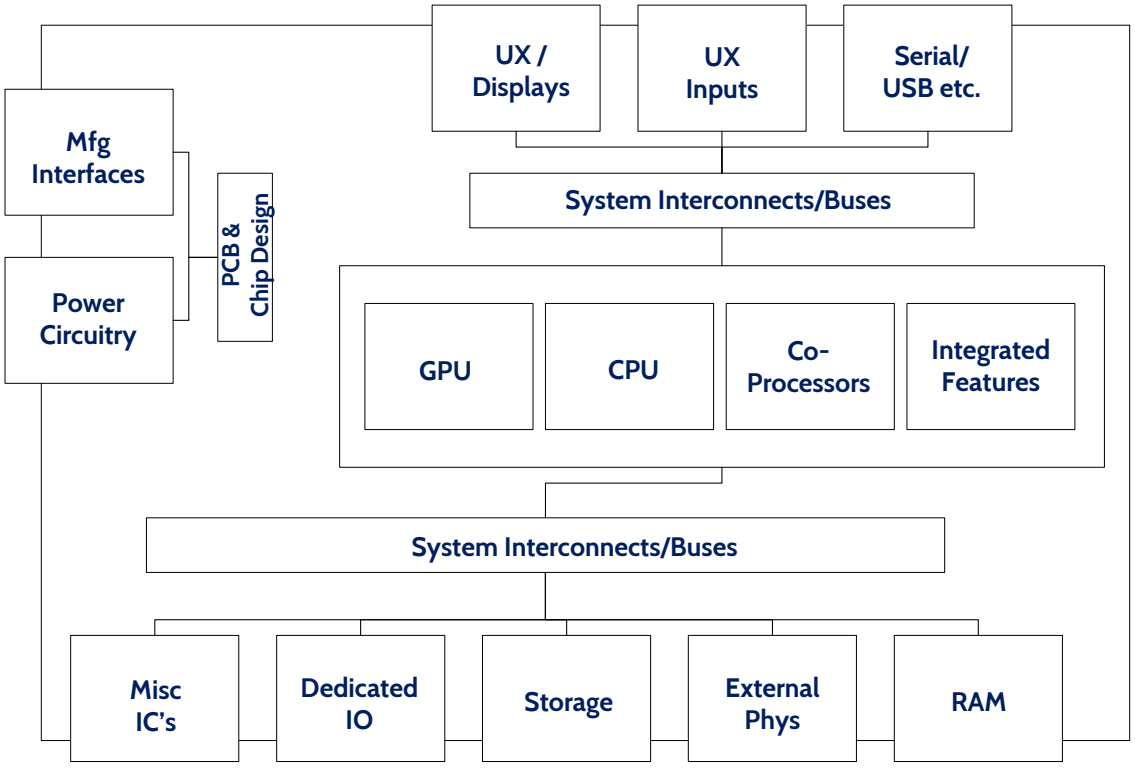

Figure 1: Physical Component Architecture Example

As you can see, there are a few basic areas that need to be understood:

- Core purpose of the system
- Hardware that encompasses the entire system
- CPU or a specialized processor/module
- Memory or RAM
- Storage for the file system, configuration or data
- Connectivity modules for networking
- Interfaces for inputs/outputs (IO)
- Interfaces to connect other chips (ICs and buses)
- Secondary or purpose-specific chips
- Power circuitry
- Software that makes the hardware operate as designed, therefore as a product

For most asset owners or consumers, this is more than enough information to understand what an embedded system is all about, but it is important enough to know how these components enable a device to operate.

Each of these components may require certain drivers, configurations, specific software to power-on the device, and bits/bytes to be sent to chips to enable functionality, and all of that + hardware/software *make the system an embedded system.*

There is a whole world dedicated to this domain, and I have left out some elements that are likely going to convolute this topic, but also on the hardware level. There are a number of interfaces often left over in final products that were used for manufacturing, accessing consoles, and even for quality assurance purposed during production. Be aware they they exist, but those are out of scope generally for most asset owners.

WHAT IS FIRMWARE?

Taking a step back now, we know that cellphones, IoT devices, and PLCs share a common ancestry – electrical engineering, hardware components, and software to "glue" it all together. However, what is unclear is how this really is different than a PC – after all, they have drivers, file systems, and applications.

Historically, PC's were a closed ecosystem, but eventually, they became more open, and easier to manage or the vendor/platform had good enough support built right in (e.g., network stacks). However, embedded solutions because of their nature to fulfill the goals of an envisioned product – are tightly controlled in both scope, development, costs, and maintenance vs. a PC where it should just work with everything. At the end of the day though, it's really about cost, and as costs go down on more capable components – embedded solutions will converge similarly to PCs, but likely with amble layers of abstraction to decouple software from hardware.

Until that day though, embedded solutions typically come in a few flavors:

- **Microcontrollers** - where all software including a "micro" OS, bootloader, applications and drivers are all in the same chip. There might be limited storage, but the key element to remember here is that these are very specific chips that focus on completing a few designated tasks under specific constraints. That's it, but they may be expanded by other peripheral chips and do not have an MMU.

- **CPUs with a MMU** – these are X86 chips, PPC, xScale, Intel, AMD, and ARM just to name a few. These chips may run a limited OS such as that as a Real-Time OS (RTOS), VxWorks, and even Linux. These systems are often far more capable and are the genetic link to the IoT devices today, or even provide mini PC-like designs such as the infamous RaspberryPi.

- **FGPA and ASIC based** - these are highly specialist solutions, and I will not get into the why or how, but they are very tightly coupled to the hardware similarly to micro-controllers and should be noted.

It's Misleading

Firmware can be both the entire files ystem or software required to make electronics "work", or it can be a specific component, and even be labeled as an update. Therefore, it is important to be clear on its scope OR its nature in general

Regardless, embedded systems have a symbiotic relationship tying hardware to software, and it is that relationship that operates the device, and that "entire thing" is called firmware collectively. Generally, firmware can be all the following, or a subset of them (which I'll break out in the next section):

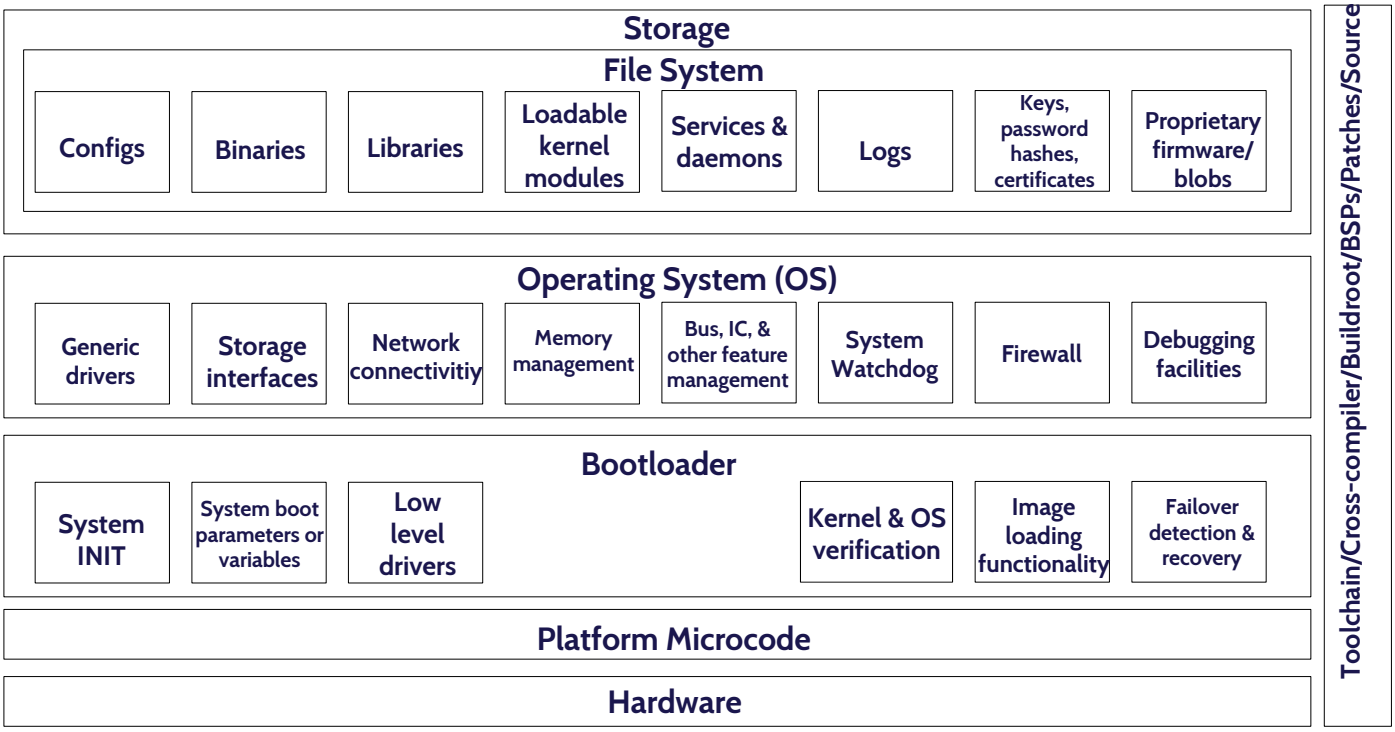

Figure 2: Simplified Diagram of a "Straightforward" Architecture

Depending on your development background, experts consider firmware a misnomer, but also as a collective term that describes everything running on a system, that's not a traditional PC-type device or even a small section of the software or a configuration required to update/alter a device. It can be "versionable", describe the platform/kernel/bootloader, and even the vendor applications specific to the product, etc.

There is a whole world in there, and we haven't even added the world of hardware-specific vulnerabilities such as exposed programming interfaces (e.g., JTAG) or even compiler and programming language-related inheritances (e.g., GLIB or GCC).

WHAT IS CONTAINED IN FIRMWARE?

At this point, I imaged this conversations went from *"I know embedded system security is going to make vulnerability management difficult,"* to *"This will be impossible if vendors struggle to write secure code, provide timely updates, and I have my own constraints such as respecting scheduled downtimes or outages."*

The reality is that updating/patching except for microcontrollers/FPGAs is unlikely to occur today unless it solves a process-specific issue. But for systems that use CPUs and commodity (also referred to as components that have out-of-the-box support for OS because they include "mainline" support) - patching as a process shouldn't be that hard if vendors/OEMs reduce their obstruction as they often are a bottleneck in the whole firmware system.

But fear not. Let's break down what is in firmware a bit more, understand these components, and some potential vulnerabilities for each using Figure 2: Simplified Diagram of a "straightforward" architecture for reference.

The table below is not exhaustive, but gives an idea of the comparison I am trying to put forward. Unless an embedded system is a purpose-driven microcontroller, an embedded system is nearly identical to a high-performance system (albeit with some alterations and customizations). Some systems do have different architectures, such as micro-kernels and several other advanced isolation techniques, but for legacy systems, very few utilize those to prevent cross-component compromise.

Component Area	Specific Subcomponent	Function	Risk Examples
Bootloader	**Bootloader itself**	Instantiates the minimum hardware to boot the system and load the kernel/mount the file system	If compromised, an attacker could load their own arbitrary code
	Environment/ config/device trees	Often provides specific information used to describe a device's hardware or even OEM information	If compromised, emergency/backdoors for diagnostics may be abused, or product key/licenses can be manipulated
Kernel	**Kernel itself**	Loads the core operating system, instantiates all necessary hardware, mounts the file system	If the kernel is replaced, compromised, or exploited, the system is often owned invisibly. It can also suffer from network stack flaws and thereby be denial of services or compromised by the network
		Drivers may be loaded in, open source, binary blobs, and/or statically compiled in. This enables a variety of functions for hardware and software	If the supply chain is compromised, or a third-party module/binary blob/ contributed module has a flaw - so does the kernel
File system	**RootFS**	Contains all of the following...	Could contain any number of vulnerabilities present in the file system and components contained within it
	Loadable kernel modules and drivers	Adds an ability for modules to be loaded at will by the kernel where necessary	If modules are unsigned, an attacker may replace a module with their own and load a rootkit on the device
	Binaries and libraries	Provides functionality, daemons, connectivity, network stacks, and more	Many embedded systems do not employ signed binaries and libraries; therefore, it is very easy to side load your own binaries unless a read-only file system is employed. Secondly, many applications use software stacks and libraries here, and those are the source of many vulnerabilities and challenges.
	Configurations	Contains necessary parameters, options to run a variety of applications, or even describe the program	If the device has an insecure configuration, insecure credentials or insecure daemons/ services, the device could be compromised
	Shell and related shells	Provides "shell" functionality on the system and a variety of binaries, and even scripting interpreters	If the device has a sell and access is required, the system can be completely owned by any number of vectors if access is obtained and/or not obstructed

Ultimately, does this realization result in additional risks I need to manage? On the surface, the answer is a staggering yes, but – do not panic. As seen in the movie Dr. Strangelove or otherwise titled "how I learned to stop worrying and love the bomb", you will soon realize that there are things we can do to reduce the collective risk for an embedded system and it's components, but also to put pressure on vendors and their suppliers to improve cyber security in the long term.

Note though that I have excluded physical security concerns about embedded hardware. This is another in depth exercise for those in the world of embedded electronics, but for an asset owner, just be aware that if someone can get physical access – they can usually compromise a system. That also includes obtaining a similar product, exploring it off-line outside your environment, weaponizing any findings, and bringing them into your environment (theoretically); fortunately, this might be more of an OEM focused threat.

HOW DOES FIRMARE FIT INTO A PRODUCT? HOW DOES IT ADD SECURITY, FUNCTIONALITY, OR LACK OF?

This circles back to the concept of firmware is largely that is simply either a complete "image" containing everything a system needs to run, or a small package of updates, configs, or binaries (even updates). It basically is the logic that makes relatively lifeless electronics come alive and perform their duties.

Therefore, firmware is an integral part of a product, but it can contain vulnerabilities, add remediations/ hardening, decrease or increase security through the provisioning of services or programs, and can also be completely devoid of any security or security-related functions.

Conversely, when a vendor states "secure firmware" – it might mean any number of possibilities. For example:

- Does it imply the firmware is encrypted? Is it signed and validated for authenticity and integrity?
- Does the word "secure" imply specific security functionality, literally embedded and used within the product? Such as tamper-resistant hardware.

Don't Forget the Hardware

Physical hardware, interfaces, and the protocols interfacing them can generate vulnerabilities or conditions that affect a devices operation. Be aware that often many different versions of the "same chip" can and do exist, but may have documented "errata" also known as flaws.

- Does it imply that the system has been hardened? Or follows a rigorous software development lifecycle (SDLC)?
- Does it mean security functionality is contained within it? Or only more secure functionality is provided, and not necessarily deployed out-of-the-box unless configured?

And should a vendor be taking proper precautions or enabling security features within a product, you the asset owner can benefit from them because a malicious entity will require additional efforts (theoretically) to compromise the device, or might even reduce/eliminate threat vectors to your environment over the long term.

Security is often fodder for the industry marketing machines, but it is important to tease out the definition of security, and to validate it. In fact, it is your responsibility as an asset owner to make an informed decision where possible, and to ensure due diligence when it comes to the security of critical infrastructure or industrial environments.

WHAT KINDS OF VULNERABILITIES CAN BE PRESENT IN FIRMWARE OR AN EMBEDDED PRODUCT?

While we flirted with the terminology of what a vulnerability is, or even the definition of a flaw – what does it mean? And types of vulnerabilities at a general level can be present within an embedded product?

In essence – a vulnerability is quintessentially a flaw, that if under the right conditions, can be exploited by user with mal-intent, result in instability or failure over time with little interaction, and/or by accident. In other words, a "bug" that results in anything, but the expected "correct" behavior.

In OT, or ICS environments, those flaws and the threats that might take advantage of them often differ from those in traditional enterprise IT. At the end of the day though, the asset owner still needs to understand what they are, how they relate to their environment, and which to remediate.

From a classic perspective, there are seven high-level vulnerability families:

- **Software vulnerabilities**
- **Hardware vulnerabilities**
- **Network and communications vulnerabilities**
- **Logic and configuration-based vulnerabilities**
- *Physical vulnerabilities*
- Organizational vulnerabilities (including deployment environments)
- Personnel-related vulnerabilities

To some degree, embedded systems can suffer from all the general areas – especially within an asset owner perspective. And on the other side of the coin, a vendor might only need to concern themselves with the threats and vulnerabilities as they relate to the product only. For the most part, an asset owner should be concerned with protecting themselves at the physical, organizational, and personnel levels regardless of the system under consideration, but to also be considering the vulnerabilities outright for a specific product such as an embedded system (items in **bold** indicate primary risks, and secondary risks in *italics* that apply to embedded systems).

And lastly, it is important for an organization to understand that they often add unnecessary risks to embedded systems by insecurely deploying devices (e.g., default or weak passwords, and insecure protocols when secure versions exist), and they often deploy insecure logic or mis-use security features. Therefore, it is critical that even PLC logic and configurations are considered from a security point of view.

WHAT ARE VENDORS DOING, AND WHERE DO FIRMWARE-RELATED PROBLEMS ARISE IN A PRODUCT?

Verve would love to say that most vendors are doing a great job, but in truth, writing software is difficult, and suffers from both the human condition & the consequences of business motivators. Devices are complex system of systems, and as they get more powerful, or offer even more features – the likelihood of a vulnerability being present also increases.

However, some vendors are doing a better job than others, and those that are quickly responding with clarity to their customers are leading the pack, and often regularly publishing security disclosures while assisting customers in finding solutions. Other than that, here are some areas that may seem obvious, but directly cause or complicate managing vulnerabilities in an embedded device:

- **If a product is end of life (EOL), insecure by default, and vulnerabilities are found in a component within it – these vulnerabilities will be carried to the device's grave.**

- **Many device vulnerabilities arise from a lack of quality and robustness testing.** Today, many asset owners are requiring audits and certifications of products, which ideally, will act as a second set of eyes on a vendor's products & practices. This can be gamed, but ideally, a company is also doing their own security testing for due diligence.

- **Some vendors have encrypted update/firmware packages, but most vendors may or may not use other security mechanisms correctly (or at all),** such as signed bootloaders, file systems, kernel modules etc. Therefore, it may be a false pretense that they offer “secure” firmware.

- **Many products improperly use cryptography primitives** and therefore, suffer from several implementation issues, or even re-use of “secrets” such as asymmetric/symmetric cryptographic keys.

- **Insecure configurations, especially defaults hinder the device’s security**, but unfortunately, this is pushed onto the asset owner by both the vendor and integrator. Alternatively, configuration options may conflict with each other and produce an insecurity, and therefore, validation is required.

- **ALL PRODUCTS contain software that is obtained from somewhere else, and this poses some statistical possibility that it can introduce a flaw that might be exploited under SPECIFIC conditions.** Without scaring the audience, a compiler may introduce flaws, a C library may have a poor implementation of malloc or a binary function, and software may rely on a software component or library obtained from another source.

- **Some software and protocols are insecure by default or design, have poor implementations, AND little can be done about it except to disable them, limit access, or use a more secure alternative, if available.** For example, if Telnet is enabled by default, but SSH is potentially available, Telnet should be disabled, and SSH used (if necessary), but this might not be possible

- **ZERO PRODUCT is intrinsically secure, even if developed entirely in-house, and can suffer from software engineering lapses, logic errors, or any other vulnerability.**

- **Hardware MAY also contain its own vulnerabilities, or flaws when used in a particular fashion.** For example, a network PHY that uses SPI and interrupts may overwhelm a CPU, and cause a software watchdog in the kernel to cause a reboot.

HOW DO I DETERMINE IF AN EMBEDDED SYSTEM HAS VULNERABILITIES?

To be fair, many organizations know their environments very well, and they have developed immense troves of expertise at the process level (sometimes even for cyber security), but not all have the necessary experience or available resources (e.g., time) to investigate if an embedded system has vulnerabilities (disclosed, undisclosed, or insecure by design).

Regardless, to find a vulnerability does not require a trained eye in all cases, but it does require a certain amount of knowledge about how systems work, how they are put together, how software is designed, and even a certain amount of detective skills. Systems engineering, programming/computer science, and even cyber security can accelerate vulnerability discovery. Initially, there are some systemic approaches to discovering vulnerabilities without high-risk/active discovery, this can happen in a couple of ways:

- **From a known and up-to-date database of vulnerabilities, compare your inventoried assets to that list.** This can include software on a host, to even embedded systems, but readily available in many product's dashboards for example.

- **Looking for information that implies legacy protocols are in use.** By themselves, legacy protocols are not a direct security risk, but unprotected and unfettered access to them is. And using that same train of thought, there is nothing wrong with them being present if they are being protected & provide essential service, but you can note them by having an appropriate product intelligently discover assets and examine the results

- **Using industry/product knowledge, look for markers of weakness.** For example, if a device has an FTP server, and it has a particular OS vendor such as VxWorks being listed on the banner – you might be able to find a series of undisclosed vulnerabilities for this particular device (e.g., by cross referencing the version provided by the OS vendor, to that in the OEM product in front of you).

- **Reading documentation such as release notes, or deployment guidelines.** Often weaknesses are noted within those documents such as, device will only accept one connection at a time, or if multiple connections are performed, the oldest will be shut down and so on…

- **Continually monitor all product, vendor, and potential third-party software component providers for updates and security commentary.** This can be fairly time consuming, and assumes you have an active and accurate inventory of your assets, but also, may be overwhelming to make sense of the jargon and challenging to determine relevancy to your environment. If this can be automated and consumable, there would be a lot of value in this approach.

The above are very safe and reliable methods to potentially discover vulnerabilities but are not "deep" by themselves. One method might be to invest in your own technical testing and follow processes outlined in ISA/IEC 62443 for example, but for more in-depth vulnerability discovery, submission of devices to researching firms, specialized analysis, and other deep knowledge can be obtained by a variety of experts or companies. However, for the curious – it generally follows an approach that looks like:

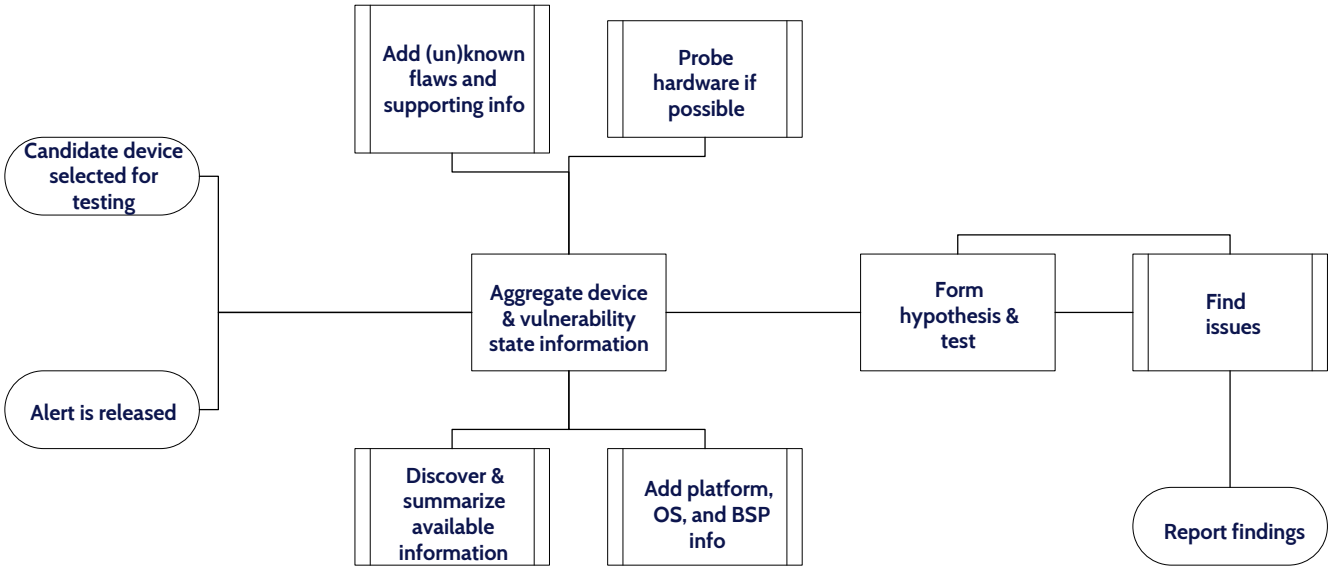

Figure 3: General Approach to Finding Vulnerabilities, but Assumes a Number of Tools, Probes or More Delicate Expertise to Obtain Adequate Confidence in Results

Unfortunately, all vulnerabilities may not be reliably found, and so it is wise to take a defensive position that assumes all devices (even with zero published vulnerabilities) are likely to contain vulnerabilities that could impact your organization. It also takes some finesse to determine their achieved protection levels, capabilities, and how to limit the likelihood of threats that may target or affect them.

HOW DO I PROTECT EMBEDDED SYSTEMS FROM VULNERABILITIES?

In a perfect world, all devices would be secure by default, or have vulnerabilities that are easily resolved. Unfortunately, this is not often the case, and surely, this series of vulnerabilities will likely persist long into the future – including after the retirement of many devices.

And after all, the ability to patch many embedded systems is dependent on the OEM, the parties that contributed components, and your organization's preferences towards updating embedded OT systems.

Therefore, neither any security product vendor nor an asset owner can outright issue firmware-related fixes in many scenarios, but what they can do – is to help you secure them (mitigate or remediate vulnerabilities) with adequate compensating controls that reflect your organization & the risks those assets might pose if compromised.

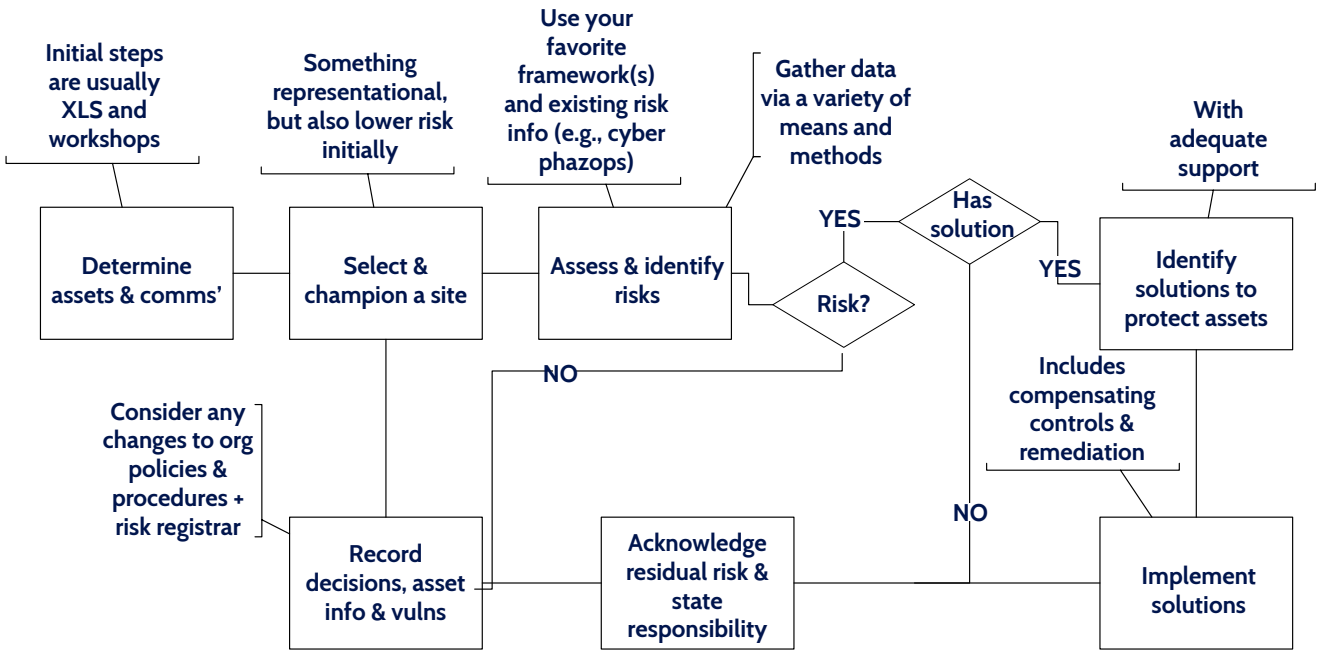

Figure 4: Another Generalized Example Process to Examine How to Protect an Embedded System (or Systems Under Consideration) with Reflection on their Deployment and/or Related Context, Specific to the Organization

Again, the good news is that an asset owner has a series of powerful tools easily accessible on the market and a number of capabilities already available to them regardless of the realities of managing some types of vulnerabilities in embedded systems. It is very doable to identify embedded systems, their models, known vulnerabilities, potential network born risks on a number of legacy insecure network protocols, weak authentication schemes, and even to determine their location (with a bit of work sometimes). And it is through these products or capabilities that you secure embedded systems through a layered approach (or defense in depth) through securing assets via their functional role, but also by their site, zone, and/or communication conduit.

To manage the risks potentially represented by embedded vulnerabilities, or to remediate them, we recommend to asset owners the following (but not specifically limited to):

- **Continually maintain an accurate inventory of deployed assets and consider embedded systems as part of your vulnerability, risk, and OT systems management programs.** Notably, these systems may have different operational constraints, however, several techniques have been successfully been used to safely identify, inventory, and manage non-commodity devices. Verve recommends including them as part of a holistic solution that raises the visibility of these devices such that they are illuminated for actionable risk management reduction and remediation.

- **Ensure firmware and software updates are obtained from a legitimate source.** This could be software directly downloaded from the product's vendor portal (e.g., X OEM portal), or confirmation that the software matches a legitimately distributed hash, which can be used to confirm authenticity, but also that it is 100% exactly as released; this could prevent any corruption during transit etc… Secondly, ensure all software and firmware is stored securely and archived appropriately within the organization.

- **Evaluate your desired security levels for the devices, their function, or zone under consideration.** This might seem obvious, but it might be fair to both assume that embedded devices need protective controls, and that they need to be securely deployed. If a device is performing a specific function over another, then another target level might be needed, or you may want to re-evaluate your deployments (and integrators delivery) to determine if secure deployment guidelines were followed. However, it has been known that security decreases over time (rots) and therefore, re-evaluating should be continuous and technologically driven if possible.

- **Patch or update affected devices that have a firmware update available WHEN/HOWEVER that is applicable, and/or re-configure devices such that they do not use vulnerable protocols where feasible.** This can potentially lower the risk exposure of a device, improve device stability, prevent accidental encounters (e.g., situations that are discovered by accident without a malicious entity) with these vulnerabilities, and deny a network-based attack vector from potentially affecting your environment. I do acknowledge though that this is at the discretion of the asset owner, and a mature risk/vulnerability management process is required to assess whether a firmware update is applicable or required (if it even exists at all).

- **Track vendor security portals through automation and mailing list subscriptions.** Of course, it should be obvious, and it might take a little bit of investment, but security updates in OT are far less frequent than those in IT, and this should be less strenuous than you might imagine, especially if partners offer services to support risk and vulnerability management.

- **Protect systems, network infrastructure, endpoints and points of ingress that are commonly compromised, have a higher risk exposure, and possess a statistical likelihood of being targeted.** A majority (likely most attacks) originate from commodity IT systems or have a significant effect on the success of an attacker, and as such, should employ a variety of recommended cyber security controls and best practices. This includes protection against commodity malware threats, security updates and patches, application whitelisting, secure configuration/hardening, firewalls, monitoring, and other technologies.

- **Prevent direct access to vulnerable devices by using network access controls, segmentation, and limiting network communications to only authorized systems and/or specific communication protocols;** in particular, isolate them from business networks and functions through zone and conduit protection concepts. This includes ensuring DNS connections are monitored, filtering upon network protocol flags, block fragmented IP traffic, block uncommon ICMP messages and responses, disable DHCP or use an alternative, disable IP tunneling, disable or block IPv6, block/sanitize if possible malformed Ethernet and TCP frames, limit to traffic only trusted domains or zones, and ensuring that these devices cannot be reached from the Internet.

- **Sanitizing network protocols for irregularities or non-standard usage by a capable network security device** may also reduce the risk or reduce unintended behavior by a device using a stack such as this one. However, it may cause issues for devices or vendors that regularly exhibit non-standard behavior and disrupt network communications.

- **Monitor ARP tables, DHCP requests, DNS lookups, IP connections, and other network related infrastructure for conditions that would enable an organization to identify malicious activity or an anomaly.** All logs should be forwarded, monitored, and investigated as part of a continuous process to manage alarms in a way to detect events, and differentiate those from accidental, steady-state, and malicious activity.

- **Ensure default passwords are changed to something of reasonable hardiness and ensure insecure legacy interfaces are disabled at deployment IF secure alternatives exist.** The chances of an organization migrating to a more secure "modern" alternative is very low in a running environment, but also CVEs or alerts are less likely to ever be released for a known security issue IF a secure option is documented/provided; therefore, vulnerabilities will NEVER/RARELY ever be noted by single focus vulnerability management solutions that rely solely on scanning.

- **Use an OT SIEM to receive asset and application logs where possible and monitor for deviances** or fluctuations in the environment. Most environments are steady-state (rarely change or are very predictable) and so identifying anomalies should be infrequent if alarming is sufficiently tuned (just as you would on an HMI or SIS). Focus on quality and context regardless.

- **Ensure multiple levels of security exist to protect embedded devices.** It is safe to assume a number of hurdles exist in receiving updates or even deploying them, but it is also safe to assume that "unknown" vulnerabilities exist within them, ranging from zero-days to "forever days". Best to protect investments with layered security and engineer risks out if possible.

- **When remote access is required to access these systems or sites home to these systems, use secure remote access technologies** with the acknowledgment that a VPN is only as secure as the connected devices (e.g., the remote workstation may be insecure and pose a threat if compromised)

- **Perform regular polling of changes on a system for unauthorized firmware revisions, configuration changes, or process logic.** This is even more true when devices have functionality that allows for localized clearing of logs or when connections are not always online to "upload" the latest copy of logic to a managing station.

- **Adequate governance, policy, and procedures to sufficiently handle risk, vulnerabilities, and incidents.** It is one thing to receive an alarm, but another to be able to act on it in a procedural and consistent manner.

- **Regularly perform "fire drills" for cyber security in ICS environments.** This includes performing forensics on systems where possible, but also the end-to-end testing of a process designed for disaster or event handling, reporting, and recovery.

- **Regularly monitor for unauthorized disclosure of intellectual property (IP) or logic, but also anything which may give an adversary an advantage.** For example, logic on Pastebin, company passwords on the dark web, GIT repos, employee LinkedIn profiles etc. After all, many embedded vulnerabilities are insecure by design vulnerabilities and all one needs to know is the default admin username, plus your organizations' favorite list of passwords (e.g., AbcOrgPassword123).

- **Leverage partners and services to provide timely vulnerability information, testing and validation for vulnerabilities and risks in your environment.** What this means is, if a device is running X, but the vendor has labeled the product as End of Life (EOL), or is being secretive about the contents within it, it might be best to perform adequate due-diligence to test for vulnerable components contained within it, and a) contact the vendor with concerns, or b) take adequate measures to reduce inherited risks, and c) report the vulnerability to relevant authorities; especially if in a compliance-based environment.

- **Request that vendors provide certification for cyber security, and validation of security properties** (or achieved security capability levels). One approach is to ensure testing as part of the conditions to satisfy the RFP, and if they cannot, be ensure that legal and risk management are astutely aware of the risks this may entail. Obtaining reference samples, validating secure deployment guidelines, and performing security testing is also another great way to improve security upfront rather than later once the investment has already "hit the floor".

ABOUT THE AUTHOR

Ron Brash is the Director of Cyber Security Insights at Verve Industrial Protection where he injects a variety of technical expertise ranging from vulnerability research to cyber-risk advisory from several critical infrastructure domains (O&G, energy, utilities, aviation). He also has over 12 years' experience working with embedded industrial control systems, possesses a MsCompSci, and a BTech.

Previously, he provided technology risk advisory, founded a successful systems design consultancy, created the S4 ICS Detection Challenges, and was an embedded developer specializing in network deep packet inspection (DPI) for industrial environments.

ABOUT THE COMPANY

With over 25 years of OT expertise, **Verve Industrial** is an industrial control systems cyber security company. Verve partners with clients to bridge IT OT security challenges in industrial environments.

The Verve Security Center provides robust asset inventory, vulnerability assessment, threat detection and the ability to safely remediate risks in a unified software-based platform.

Verve Industrial serves industries across utilities (such as power, oil & gas, water), manufacturing, healthcare and building controls.

Please visit us at www.verveindustrial.com to learn more.

INFO@VERVEINDUSTRIAL.COM
888-756-3251