Literature Review of Embedded Software Vulnerability Protection and Mitigation Schemes

Nathan Palmer
Department of Electrical and
Computer Engineering
Mississippi State University
Starkville, Mississippi 39762
Email: ntp1@msstate.edu

***Abstract*—Embedded system software as a class is very dynamic and expansive, both in reach of applications and in variety of implementations. However, there are some properties that make it particularly vulnerable to certain types of attacks. The characteristics, and notably the constraints, of embedded systems cause challenges for protection of critical infrastructure, personal health, automotive, logistics, and many other systems integral to daily life. Research on these specific challenges is rapidly changing and not as mature as more general software security research. This paper outlines the state of the literature regarding embedded software security and provides a list of mainstream computer security trends that should be researched in the context of embedded systems.**

I. Introduction

A high degree of security in embedded computer systems, specifically those used in life critical or biomedical devices, is a particularly important design goal. Embedded systems are designed for hostile environments and operate with uniquely constrained resources. This makes embedded systems a vulnerable class of software that requires a specific set of security processes and guidelines that has not been well defined.

There is ample research on effective software security practices for general purpose computing platforms and it is likely that many of those practices can also be applied to embedded systems. This paper provides a systematic review of recent literature related to embedded software security and general computer security trends. Issues specific to embedded software are described and prioritized based on common security risk analysis metrics. Special emphasis is placed on wireless sensor networks and therefore literature related to low power and resource constrained wireless security will also be included. Also, general security trends that may apply to embedded systems but seem to be under researched by the literature are presented as possible areas of future research. The goal of this paper is to collect and report the security metrics and methods that have driven the development of secure system and application software which may also apply to embedded software.

Security researchers and practitioners can utilize these results when designing, targeting, or interfacing embedded software and platforms. Those readers with experience in general software security practices gain perspective on where embedded systems fit into the broader security landscape. Embedded software engineers are presented a review of security best practices and references for use during implementation.

The rest of this paper is organized as follows. Section II: Background and Related Work, provides a background on embedded systems and the general security challenges that affect the design, implementation, and maintenance of software running on those systems. Section III: Methodology, details the literature search, selection, and filtering process that was followed while performing this review. Section IV: Analysis and Results, contains the results of this review organized by Common Vulnerability Scoring System (CVSS) score and reference count. Section V: Conclusions, provides a summary of the lessons learned and includes areas for further research.

II. Background and Related Work

Miniaturization of computing hardware continues to drive the expansion of software into more and more areas of our lives. This notably includes critical infrastructure, personal health, transportation devices, and diagnostic tools. The software that resides on devices such as these provide control and communication interfaces for the underlying hardware. The hardware that is included in

these systems varies significantly between application domains but some generalizations can be made. The diversity of targeted devices is one of the reasons why embedded software security is particularly difficult.

A. Embedded Systems

Embedded systems are characterized by tight integration and coupling with specific purpose hardware. This is in contrast with application and system software developed for general purpose computing platforms such as racked servers or laptop and desktop computers. Embedded software is used in cases where general purpose software is not available due to application constraints. Typical constraints that lead to an embedded software solution are low cost, low power, and limited energy. Developing software under constraints such as these limits the resources available for security primitives such as encryption, redundancy, privilege management, and others. Filtering specific security primitives and practices that are suitable for embedded software is an under serviced research area and one of the goals of this paper.

The following section lists some of the security related characteristics of embedded systems and software:

- Timing constraints are very tight. Real-time operating systems (RTOS) tend to eschew security for low latency. Many systems have custom operating systems that are written for a specific task and does not gain the benefit of a time-tested operating system. Also, denial of service (DOS) attacks can be particularly effective due to minimum timing margins. [1]
- Device drivers and protocol stacks are often developed in an ad-hoc and custom fashion. This means that they may not be maintained and updated with upstream patches the way traditional systems could be.
- The systems themselves are often in hostile environments. The software, hardware and interfaces are all in the attackers hands which makes reverse engineering easier than cases where the software is running on a remote server. Embedded systems must be designed to be tamper proof to prevent modification or discovery of sensitive data.
- Embedded systems tend to have low power and computing overhead which makes any security features a hard sell. Business processes may not be in place to properly prioritize software security. Low energy, battery powered systems such as implantable or remote sensors are also subject to denial of service attacks involving energy draining. An increase in the duty cycle of high power features could lower field life these systems. [2]
- Distributed embedded systems, such as wireless sensor networks, often operate in hostile environments where communication channels (wired and wireless) cannot be considered secure. This necessitates encryption for command, control and communication. However, proper encryption may not be available on the embedded platform in use. [3]

B. Approaches to Embedded Security

For many years high security was not seen as essential for these types of devices because physical barriers could be placed around the computing hardware and there were few interfaces by which a would be attacker could access the system. This is no longer the case. Current embedded systems require networking and configuration interfaces that are, many times, user accessible. This is especially true for network enabled devices. These contain many of the same vulnerabilities as traditional web servers but often lack the robust security emphasis that is afforded traditional web servers. [4]

C. Wireless Sensor Networks

Wireless sensor networks (WSN) comprise a special class of embedded system that is defined by its connectivity. WSN devices are low power, battery powered devices that are designed to take information from their environment, perform operations on data, optionally provide feedback into the environment, and communicate with other devices. Typically, these devices form dynamic and adaptable (ad-hoc) networks among the various sensors and may perform distributed processing on the data as well.[5] WSNs are also notable for their high profile applications such as safety critical environmental monitoring, implantable health monitoring (mHealth), perimeter security and inventory tracking.[3], [6], [7] Distributed WSN topologies are very different from traditional computer networks, such as the Internet, which tend to centralize services to specific nodes. However, this paper discusses areas of software security research that, while not originally targeted at distributed designs, can increase the security of wireless sensor networks.

III. Methodology

The main goal of this study is to provide the reader with software security practices that have been reported by literature to apply well to embedded system software design and development. However, there are several sub-goals that aid in meeting that objective.

First, a high level study of the field is performed. This is mainly a non-academic review of literature from trade magazines and online sources. The purpose is to gain an understanding of the field such that current trends and relevant terminology is properly included in the search methodology.

Next, the resources used for primary sources is defined. This includes journals and databases that are used to find the raw data for analysis. These sources are all peer-reviewed academic journals.

Following that is the creation of the search methodology. This includes the actual search strings which are used to find journals or other articles which make up the primary sources. Terms are chosen in an attempt to limit the search space to the overlap of software security and the environmental requirements or constraints of embedded systems. This is an iterative step that is refined during analysis.

Once the search strings are created, the criteria for inclusion and rejection is also defined. These criteria act as a gatekeeper to filter results such that only relevant and high quality studies are included in the analysis.

Finally, the sources which are selected for analysis are used to synthesize an authoritative list of software security practices for embedded systems.

A. Research Questions and Hypotheses

The hypothesis this study hopes to support is that many existing software security practices can be successfully applied to embedded software to increase security in embedded systems but are not included in the literature targeting embedded system research. The goals listed in the previous section outline the approach to confirm this hypothesis. A further research question that is addressed is: "Where are the areas in the field of embedded software security that are high priority but under researched". The following section provides the results of some of those goals and a detailed plan for performing the final analysis.

B. Current Embedded System Trends

Current trends in embedded software development are summarised in this section in an effort to provide a context for a more thorough literature search and also to define some relevant terminology. Most of this information was obtained from the trade publication Embedded Systems Design, the blogging portal site Embedded Gurus, and Wikipedia.

The most obvious trend in the field of embedded systems is increased connectivity. The so-called "internet of things" is seen by many as the next iteration of the internet. Instead of being populated by data and content entered by humans, like the current internet is, the "internet of things" is made up of physical assets that provide streams of data (environmental, biomedical, logistical, etc.) from smart sensors. 25 billion such embedded software devices are expected to be connected by 2015. Largely due to this prediction, more sophisticated networking protocols and hardware must be included on next generation embedded systems. One example of this is IPV6 network addressing which was developed to increase the maximum number IP addressable internet nodes above 4.3 billion unique addresses.

Another trend in embedded systems is wireless connectivity. CPU and battery technology have progressed to the point where it is feasible to include wireless communication features to small, low power, low energy, and low cost sensors and controls. Examples of common wireless technologies include Zigbee, Bluetooth (BTLE), and 802.11 WiFi. Each of these protocols have specialized profiles for low energy and medical applications.

Network topologies are also an important research area for embedded systems. Wired computer networks tend to be star, ring, or bus type networks. However, there are advantages for wireless embedded systems that operate on mesh networks. Specifically, a dynamically managed mesh network of wirelessly connected nodes can lead to a more robust and fault tolerant network (See Fig.1). Also, the network can be self-assembling. This makes it easier to deploy networks of sensors into environments where it is difficult to predict exactly where nodes will be placed and which nodes will ultimately be connected to one another.[2]

Another trend in embedded system design is to include field upgradeable firmware. Along with connectivity comes the desire to provide updates and patches to deployed embedded software (firmware). This functionality can apply to individual applications and features

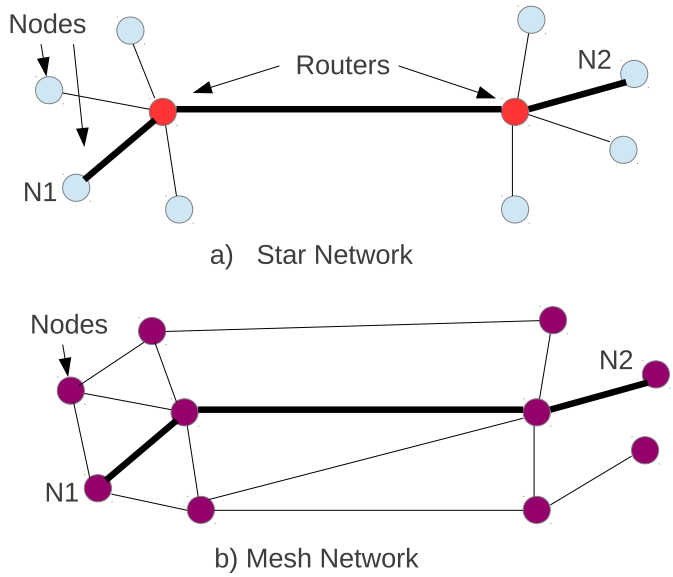

Fig. 1. Examples of common bus topologies connecting two nodes, N1 and N2. a) A pair of star networks connected via a pair of routers. b) A mesh network where every node can also act as a router. New routes can be formed dynamically if one node leaves the network.

or to the entire software suite on the device, including device drivers and system software. Depending on the implementation, updates may require user intervention or may be pushed to the device from a vendor. Ensuring the integrity of field installed firmware is an active area of research.

Traditionally, the majority of development for embedded software was done in low-level languages such as C or in architecture specific assembly code. While these languages are still dominant, there is a trend towards using higher level languages for all but the lowest level functionality. Some popular languages taking a foothold in embedded software development are Embedded Java and Embedded C++. Features such as type safety and garbage collection are new to many embedded software engineers.

Consolidation of functionality is another feature of emerging embedded systems. Many embedded systems consist of a single system-on-a-chip (SoC) that includes not only the CPU, RAM and ROM components, but interface hardware as well. Many include hardware for efficient network connectivity such as WIFi, Ethernet, NFC/RFID, cellular and Bluetooth physical layer devices. Also popular are USB layer hardware, encryption accelerators, analog to digital converters, and digital signal processing (DSP) hardware.

C. Traditional Embedded System Features

Many traditional features of embedded systems continue to affect current software design and implementation. A reduced energy budget continues to be a constraint for battery powered devices and leads to other constraints such as low processor clock speeds and reduced system memory. Less processing power reduces options for for security features such as encryption. Also, it continues to be difficult to provide general purpose embedded software because the code is heavily dependent on the underlying CPU platform. Platforms can vary in memory architecture, word size, instruction set, floating point support, and register sets. Real-time performance (guaranteed bounds on latency) constraints are still a requirement of many embedded systems. Finally, debugging and testing is complicated by the lack of an underlying general purpose operating system and the sensitivity of the system to timing variations. It often requires the use of sophisticated software emulators and simulators. Often specialized hardware is also required to debug or profile the system.

D. Primary Sources

Peer reviewed journals provide the raw data for the analysis included in the following section of this paper, Section 4. Access to the EBSCO Host database provides the searching capability utilized to find candidate papers. All peer reviewed journals are included in the search space, but the publication date is limited to the years 2009 through 2013.

Two sets of search results are included for analysis: embedded software security specific articles and general software security articles filtered by key constraints. The following search criteria provide the articles under review.

Embedded Systems Software security

Find Articles where the abstract contains:

(embedded AND software) AND (security OR secure) AND (metric OR protect OR attack OR vulnerability)

General Software Security with Constraints

Find Articles where the abstract contains:

software AND (security OR secure) AND (”real-time” OR mesh OR ”low power”) NOT (embedded)

E. Selection / Rejection Criteria

Literature returned by the previous search criteria is further filtered based on the relevance of its results to the research questions defined above. For the *Embedded Systems Software Security* results this process is accomplished by reading the abstracts and selecting any papers that show improved security due to novel protection schemes. For the *General Software Security with Constraints* results, the papers are filtered by reading the abstract and comparing the scope of the study to embedded system criteria and trends as they are described earlier in this section. Any papers that show relevant protection schemes are included in the final analysis.

F. Analysis Methodology

Each paper that is selected in the previous step is analysed for protection schemes and vulnerabilities. A table of vulnerabilities is included in the analysis section that includes a brief description and the number of unique papers that reference the vulnerability. A score is assigned to each vulnerability that represents its relative impact to system security. The score is based on the Common Vulnerability Scoring System (CVSS) defined by the National Institute of Standards and Technology[1]. Any protection or mitigation schemes that are reported in the literature under review are provided for the applicable vulnerabilities listed in the table.

IV. Analysis and Results

A. Embedded Software Vulnerabilities

Table I gives a representative list of vulnerability types found in the current literature. The types are defined as follows. System software attacks are those that exploit a weakness in the underlying operating system or device drivers. This could be an upstream flaw that affects all instances of the operating system or it could be specific to the particular implementation. Application software attacks exploit weaknesses in the application specific code. Side channel attacks are those that utilize communication channels that are not intended for communication to leak information or inject data. Interface attacks exploit vulnerabilities in external interfaces such as USB and TCP/IP protocols, data collection inputs, or external memory interfaces. Finally, software modification attacks change program instruction or data memory either partially or wholly to change the behaviour of the system. The following sections go into more detail on each of the attack types and provide mitigation advice based on the current literature.

[1]See http://nvd.nist.gov/cvss.cfm for more information on the NVD Common Vulnerability Scoring System Support V2

TABLE I. Summary of Vulnerabilities

Vulnerability Type	Refs	CVSS Parameters	Score
Sys. Software Attack	3	AV:L/AC:H/Au:S/C:C/I:C/A:C	6
Software Modification	5	AV:L/AC:H/Au:S/C:C/I:C/A:C	6
Interface Attack	3	AV:L/AC:H/Au:N/C:N/I:C/A:C	5.6
Cross Channel Attack	1	AV:N/AC:H/Au:S/C:P/I:P/A:P	4.6
App. Software Attack	4	AV:L/AC:M/Au:S/C:P/I:P/A:P	4.1
Side Channel Attack	3	AV:L/AC:H/Au:N/C:C/I:N/A:N	4

1) System Software Attacks: System software controls access to hardware resources through memory management and/or device drivers. It also arbitrates access to the processor by allocating time slices for processes. Many embedded system developers write custom system software to access custom hardware or to meet strict real-time requirements. There are also operating systems that are written specifically for embedded systems. Flaws in these systems can be exploited by malicious users to compromise an embedded system.

Attacks on system software can be carried out by exploiting traditional weak links in software design, such as buffer overflows or direct memory modification. However, because system software tends to run with elevated privilege and has direct access to memory, system software vulnerabilities are more dangerous than in application software.

Most mitigation strategies suggested by the literature bypass the system software. One approach is to encapsulate integral security software, such as encryption or authentication methods, in secure memory and only access the functionality through special hardware-separated memory.[8] This allows some assurance of the integrity of the system while still allowing the use of a monolithic OS. Note that these approaches require hardware monitoring of the secure memory because it is assumed that if the system software is compromised then the attacker has full read and write access to the system's memory.

Another approach to preventing this attack is to include a separate operating system for security critical processes. A custom micro-kernel could be included to provide security related functionality to the system. This OS could execute in parallel with the main OS using hardware memory separation or it could have it's own independent memory space and communicate though a

secure bus.[9]

2) Application Software Attacks: Application code is another common area of attack in embedded systems. Exploits on application code may not compromise the entire system like system software does, but it can reveal sensitive information or compromise specific system functionality. Application code can be attacked with injected code or buffer overflow attacks among others. The literature seems to focus on two mitigation approaches to application attacks: detection and prevention.

Attacks can be detected using a combination of static code analysis and real-time monitoring. The expected number of instructions or clock cycles between specific checkpoints in the code can be determined at compile time and monitors (hardware or software) can ensure that the expected number of clock cycles or instructions were actually executed. A discrepancy indicates that the code was modified. Prevention of such attacks can be implemented via program and data encryption. The program instructions can be encoded using an encryption scheme that ensures that modifications are detected. The instructions are decoded by the processor at execution time. Modification of the code results in invalid processor instructions. [10] [11]

3) Side Channel Attacks: Side channel attacks are those that utilise interfaces to the device that are not intended for communication such as power supply, electromagnetic, or acoustic channels. Many papers have shown that careful analysis of timing, electromagnetic emissions, or energy consumption can reveal the internal state of an embedded system. [12] This leak of information can be exploited to discover key features of an embedded system such as encryption algorithms and even private keys. Power analysis attacks are a very active area of research. Power analysis attacks sample the instantaneous current flowing into the hardware (e.g. a security co-processor containing an encryption private key) during various inputs. Patterns discovered in the power analysis can be used to guess the number of rounds used an TripleDES implementation, for example. Also, differential analysis can be performed by creating a template signal that is based on an known state. Comparisons are made to this template to determine the value of unknown states. Fig.2 shows an example setup an attacker may use to reveal the cryptographic private key from a security processor. [13]

The effectiveness of such attacks can be mitigated in two ways: masking and hiding. Masking the side-channel

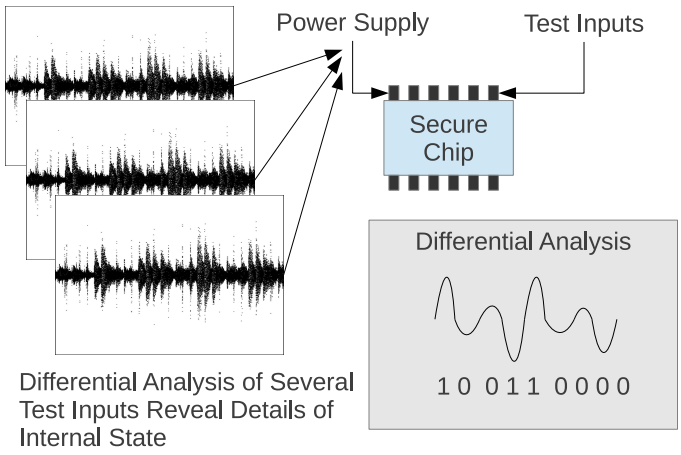

Fig. 2. Differential power analysis attack. Several measurements of supply current are made while varying chip inputs. The differences between the measurements can reveal clues about the internal state, including private security keys.

signal is accomplished by inserting random instructions to mask the signal in random noise. Hiding works by providing a consistent view of the system, one that does not change with the state of the software, to an attacker via specialised hardware such as dual-rail pre-charge (DRP) logic gates.[14]

4) Cross Channel Attacks: Cross channel attacks exploit a flaw in one communication channel to expose a vulnerability in anther channel. This is not specific to embedded systems but they are particularly vulnerable. Many embedded systems use a collection of third-party software to enable networking features for services such as media access, web based configuration, wireless device discovery, etc. Due to constraints on embedded systems, these services may have to share resources such as memory or disk space. A vulnerability in one service, such as an insecure FTP user account, can lead to other attacks such as inserting malicious Java Script code onto a device configuration website. This is very similar to a cross site scripting attack. [3]

There is no specific protection against this class of attacks. However, most of the exploits that fall in to this category also exist on desktop and server platforms so traditional security advice applies. Input from all channels should be sanitized to protect against buffer overflow attacks and code injection. Also, if possible, programmers should include the ability to patch any third party applications with security updates. [3]

5) Interface Attacks: Interface attacks are those that exploit some weakness in an external interface to a

device. Embedded systems do not typically have screens and keyboards like personal computers and many do not have a network interface but there is always an interface to the environment.

Some devices monitor sensors via probes that measure electrical values such as voltages and provide feedback to actuators such as motors or solenoids. An attacker can carefully craft an electrical signal to inject arbitrary values through inputs. This approach can be used to perform buffer or integer overflow attacks. Data sanitization should be applied to these types of inputs.[15]

Another interface common to embedded systems is external memory. Data stored on a separate chip from the processor is exposed to eavesdropping on the memory interface bus. This is commonly exploited to capture encryption keys during private key exchange or distribution. Sensitive data should never be passed unencrypted across an external bus, especially private keys. Encryption and decryption should only be performed on the device that needs access to the sensitive data.[16]

Finally, the Universal Serial Bus (USB) is a ubiquitous interface for embedded systems. These ports allow high throughput data transfers to and from the device and also allow control via USB keyboards and mice. However, the USB protocol stack is very complicated and there are many know exploits for USB drivers. Many times these drivers run with elevated privilege allowing code injection from USB attacks to cause substantial damage. The best way to mitigate this type of attack is to only provide user level USB support without elevated privilege. [17]

6) Software Modification: The ability to make field updates to firmware is a powerful feature and can be used to increase the security of a system by patching vulnerabilities as they are discovered. However, having the ability to change the code running on an embedded system should be weighed against the risk of an attacker modifying the firmware to execute arbitrary code. This type of attack is common in consumer electronics (e.g. digital cameras and TCP/IP routers) where features are enabled on low-end models that were only intended for high-end versions. Smart-phones and set-top boxes are also common targets where attackers remove key security checks from vendor's firmware and release the 'patched' versions. [11]

Protections against software modification focus on securing the software that loads the firmware into memory at the beginning of execution. This boot loader should ensure that the code it is loading is unmodified or digitally signed to ensure the integrity of the instructions and data. Another option is to only allow parts of the firmware to be writeable and keep critical program code in read only memory (ROM). Finally, code can be analysed at runtime to ensure that the code is not modified after being loaded into memory. This can be done with special hardware that counts instructions or CPU cycles between check points in the program flow to ensure that instructions were not added or removed. [10], [18]

TABLE II. SUMMARY OF GENERAL SOFTWARE SECURITY TRENDS

Software Security Trend	Reference Count
Security Model Checking	7
Encryption	5
Real-Time Security Monitoring	6
Securing Data Channels	3
Data Integrity	3
Virtualization	2
Other	2

B. General Software Security Trends

This section explores some of the recent trends in software security that have been shown to apply to low power and wireless mesh networks systems. These trends also apply to many embedded systems and are included here to provide stimulate research in specific applications to embedded systems.

Fifty seven (57) papers were discovered using the search methodology described in Section III. From those, a total of twenty-nine (29) papers were analysed to compile the following list of topics, summarised in table II. The first column names the category and the second column give the corresponding number of recent papers that focused on that topic. Literature was discarded if the abstract did not indicate an actual application to embedded systems. The following sections detail the trends and how they apply to embedded systems.

1) Security Model Checking: Models that represent algorithms and software specifications can be created by designers using formal logic. If the models are complete then equivalence can be verified between the specification and the algorithm. Code can also be checked against the model to prove that it properly meets the specifications. This type of checking is difficult due to the effort involved in creating and maintaining the formal

models but it is necessary for high assurance software. Such model checking is a heavily researched area of computer science.

Software faults are discovered in the algorithmic models before they can affect actual systems. Model checking can be a powerful tool against malicious attacks because these faults can easily lead to security vulnerabilities. Models can be created for the entire system or only for critical subsystems such as encryption algorithms. Communication protocols and other interfaces can be modelled and simulated attacks can be performed on the models. [19]

Security models for embedded systems software (including specific time constraints and physical features) and attacks should be further researched.

2) Encryption: Encryption is the standard software solution to ensuring confidentiality and often for providing integrity. Embedded systems are highly susceptible to attacks on confidentiality of data because they tend to be small, mobile, and often physically accessible to attackers. However, the resource limitations of embedded systems prevents the use of encryption for many applications.

Technologies such as security coprocessors provide hardware solutions by providing specialized cryptographic accelerators for common algorithms such as DES and Elliptical Curve Cryptography (ECC). ECC is particularly attractive to embedded systems because it requires fewer instructions than comparably secure algorithms. [20]

More research is necessary to develop encryption techniques that are applicable to embedded systems.

3) Real-Time Security Monitoring: Many attacks can be detected by their side effects to the affected system. Malicious code such as viruses and worms can often be detected by their signatures, specific strings of data that are contained in their code or data. On desktop systems, virus scanners can scan memory and disk space for the presence of malicious code. Other, more reactive, types of analyses can be performed using forensic software that looks for signs of an attack in logs or system data.

These types of monitoring approaches could also apply to embedded system software. Software similar to a virus scanner could scan firmware for changes indicative of an attack. This could be a different signature than in a desktop system, so more research is necessary. However, some work has been done on wireless area network (WAN) monitoring that is applicable to embedded systems. [21]

Embedded systems could also be analysed for anomalies in behaviour by external or redundant systems. Anomalies could manifest as behaviour that original specifications did not intend or could include checks for incorrect states, such as valves in the open and closed position at the same time or several neighbouring systems having drastically different system times or temperature readings. [22]

4) Securing Data Channels: Embedded systems suffer from the same types of issues as other computing systems when they communicate in networks over hostile communication channels. Vulnerabilities in common IP routing or framing protocols apply to embedded systems if the vulnerable protocols are in use on the network. Some technologies that are common for wireless mobile networks such as 802.11 WiFi, DSDV (Destination sequenced distance vector), and WRP (Wireless Routing Protocol) are also common in embedded systems. Attacks on these protocols should be understood by embedded developers that utilize them. [23], [24]

Also, issues such as secure key distribution in wireless networks apply to embedded networks as well. Research on secure private key distribution in distributed networks is ongoing and should be expanded to include issues specific to embedded systems.

5) Data Integrity: Software integrity refers to the level of confidence that data and code have not been modified unintentionally. Much research has been conducted on how to ensure data integrity in software. Digital signatures are the standard technology for ensuring integrity. However, the same embedded system limitations that create problems for encryption also affect digital signatures. [25]

External hardware such as trusted platform modules (TPM) can be applied to embedded systems to enable digital signature verification on executed code. This technology is used by PCs to ensure the integrity of the system BIOS and some operating systems also use it to verify critical code.

6) Virtualization: Virtualization technologies enable software isolation by providing an abstraction layer around hardware resources. At the desktop level this is usually a hypervisor or virtual machine that allows an OS to be installed to an abstraction of the underlying

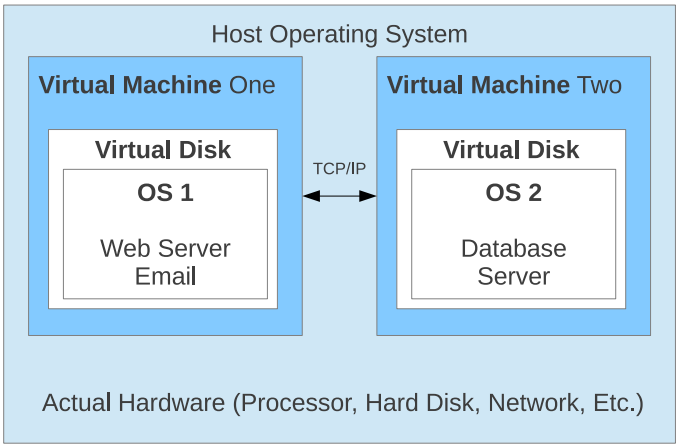

Fig. 3. Two virtual machines on one computer can provide isolation between different services. If one virtual machine OS is compromised the other is not necessarily affected.

hardware, as shown in Fig.3. Components such as the video card, sound card, hard drives, and network interfaces are all mediated via the virtual machine. Modern processors even include special features that efficiently separate virtual machine processes. [26]

The advantage to having a single machine running several virtual machines is that it enables isolation at a lower level than an operating system can provide. A web server can be running on the same physical machine as a database server without the concern that a vulnerability in the web server, even at the OS level, will lead to a compromise of the database server. [27]

Embedded systems could benefit from this approach. Embedded systems have traditionally been comprised of several separate digital devices that provide specialized functions such as signal conditioning, communications, and data processing. Newer systems use a single processor that consolidates this functionality. Software should be designed such that systems are isolated as much as possible and embedded virtualization could enable that.

7) Other: This section describes other security trends that could apply to embedded systems.

High level programming languages such as Java or even interpreted languages such as Python could be used in embedded systems. High performance and driver software still needs to be implemented in low-level languages like C or assembly. However, high-level languages tend to be less error prone and more easily prevent attacks on memory.

Also, many mobile systems use mobile tracking and detection systems to monitor the position of assets. Miniaturization of GPS and RFID technology allows this to be done for embedded systems as well. [28], [29]

V. Conclusions

There are many areas where embedded software security falls short. Many of the attacks that are performed on embedded systems are very different from the types of attacks performed on traditional infrastructure systems. However, the learning from decades of software security research may apply to these new attacks.

This paper shows that there is ongoing research on the security concerns specific to embedded software. Many new attacks are defined and mitigation strategies discussed. Side channel attacks, such as power and timing analysis, are novel forms of attack on embedded systems and processing resource constraints restrict the mitigation options.

It is also shown that there are many areas where new research must be conducted to examine the ways that traditional security techniques can be successfully adapted to embedded software. Areas such as encryption, modelling and virtualization are all staples of software security literature but must be re-examined for embedded systems.

References

[1] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Systems with Applications*, vol. 38, no. 4, pp. 4626–4636, Apr. 2011.

[2] E. Y. Vasserman and N. Hopper, "Vampire Attacks: Draining Life from Wireless Ad Hoc Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 2, pp. 318–332, Feb. 2013.

[3] H. Bojinov, E. Bursztein, and D. Boneh, "The emergence of cross channel scripting," *Communications of the ACM*, vol. 53, no. 8, pp. 105–114, 2010.

[4] G. Kumar, M. Rai, and G.-s. Lee, "Implementation of Cipher Block Chaining in Wireless Sensor Networks for Security Enhancement," *International Journal of Security and Its Applications*, vol. 6, no. 1, pp. 57–72, 2012.

[5] S. Singh, "Security For Wireless Sensor Network," *International Journal on Computer Science and Engineering (IJCSE)*, vol. 3, no. 6, pp. 2393–2400, 2011.

[6] S. Mahdavi-Hezavehi, M. Galster, and P. Avgeriou, "Variability in quality attributes of service-based software systems: A systematic literature review," *Information and Software Technology*, vol. 55, no. 2, pp. 320–343, Feb. 2013.

[7] P. K. Sahoo, "Efficient security mechanisms for mHealth applications using wireless body sensor networks," *Sensors (Basel, Switzerland)*, vol. 12, no. 9, pp. 12 606–33, Jan. 2012.

[8] D. Arora, N. Aaraj, A. Raghunathan, and N. Jha, "INVISIOS: A Lightweight, Minimally Intrusive Secure Execution Environment," *ACM Transactions on Embedded Computer Systems*, vol. 11, no. 3, pp. 60–80, 2012.

[9] W. Hu, T. Chen, Q. Shi, G. Wang, N. Zhang, J. Ma, and Y. Lian, "A Novel Operating System on Chip with Information Security Support for Embedded System," *Journal of Software*, vol. 4, no. 10, pp. 1053–1060, Dec. 2009.

[10] O. Gelbart, E. Leontie, B. Narahari, and R. Simha, "A compiler-hardware approach to software protection for embedded systems," *Computers & Electrical Engineering*, vol. 35, no. 2, pp. 315–328, Mar. 2009.

[11] K. Patel, "Architectural Frameworks for Security and Reliability of MPSoCs," *Very Large Scale Integration Systems*, vol. 19, no. 9, pp. 1641–1654, 2011.

[12] Z. Chen, A. Sinha, and P. Schaumont, "Using Virtual Secure Circuit to Protect Embedded Software from Side-Channel Attacks," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 124–136, 2013.

[13] J. a. Ambrose, R. G. Ragel, and S. Parameswaran, "Randomized Instruction Injection to Counter Power Analysis Attacks," *ACM Transactions on Embedded Computing Systems*, vol. 11, no. 3, pp. 1–28, Sep. 2012.

[14] A. Rogers and A. Milenković, "Security extensions for integrity and confidentiality in embedded processors," *Microprocessors and Microsystems*, pp. 1–28, 2009.

[15] K. Jyostna and V. Padmaja, "Secure Embedded System Networking: An Advanced Security Perspective," *International Journal of Engineering Science*, vol. 3, no. 5, pp. 3854–3862, 2011.

[16] R. Vaslin, G. Gogniat, J.-P. Diguet, E. Wanderley, R. Tessier, and W. Burleson, "A security approach for off-chip memory in embedded microprocessor systems," *Microprocessors and Microsystems*, vol. 33, no. 1, pp. 37–45, Feb. 2009.

[17] D. V. Pham, A. Syed, and M. N. Halgamuge, "Universal serial bus based software attacks and protection solutions," *Digital Investigation*, vol. 7, no. 3-4, pp. 172–184, Apr. 2011.

[18] C. Basile, S. Di Carlo, and a. Scionti, "FPGA-Based Remote-Code Integrity Verification of Programs in Distributed Embedded Systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 2, pp. 187–200, Mar. 2012.

[19] S. Ouchani, Y. Jarraya, O. A. Mohamed, and M. Debbabi, "Probabilistic Attack Scenarios to Evaluate Policies over Communication Protocols," *Journal of Software*, vol. 7, no. 7, pp. 1488–1495, Jul. 2012.

[20] X. Huang, D. Sharma, S.-I. Ao, H. Katagir, L. Xu, and A. H.-S. Chan, "Elliptic Curve Cryptography with Security System in Wireless Sensor Networks," *IAENG Transactions on Engineering Technologies*, vol. 5, pp. 519–531, 2010.

[21] S. Hwang and D. Yu, "Remote monitoring and controlling system based on zigbee networks." *International Journal of Software Engineering and its Applications*, vol. 6, no. 3, pp. 35–42, 2012.

[22] A. Bondavalli, F. Brancati, A. Flammini, and S. Rinaldi, "Master Failure Detection Protocol in Internal Synchronization Environment," *IEEE Transactions on Instrumentation and Measurements*, vol. 62, no. 1, pp. 4–12, 2013.

[23] M. Kotha, "A Unique Wireless Device Fingerprinting Technique for Secured Data Communication in Wireless Network," *International Journal of Computer Applications*, vol. 43, no. 20, pp. 14–19, 2012.

[24] U. . . . Sagar, M. . . . Waheed, and K. . . . Kumar, "Ensuring data storage security in tree cast routing architecture for sensor networks." in *AIP Conference Proceedings*, vol. 1298, no. International Conference on Modeling, Optimization, and Computing, ICMOC 2010, (1)Dept of Computer Science and Engineering, KBNCE, 2010, pp. 639–644. [Online]. Available: https://login.proxy.library.msstate.edu

[25] V. Jaglan, S. Dalai, and S. Srinivasan, "Enhancing Security Of Agent-Oriented Techniques Programs Code Using Jar Files," *International Journal on Computer Science and Engineering*, vol. 3, no. 4, pp. 1627–1633, 2011.

[26] H. Chun-Hsian, H. Pao-Ann, and S. Jih-Sheng, "Model-based platform-specific co-design methodology for dynamically partially reconfigurable systems with hardware virtualization and preemption." *Journal of Systems Architecture*, vol. 56, no. Design Flows and System Architectures for Adaptive Computing on Reconfigurable Platforms, pp. 545 – 560, n.d.

[27] T. June, "Security : The New Frontier," *Engineering and Technology*, no. June, pp. 80–84, 2011.

[28] B. Torres, Q. Pang, G. Skelton, S. Bridges, and N. Meghanathan, "Integration of an rfid reader to a wireless sensor network and using it to identify an individual carrying rfid tags." 2011.

[29] H. Kwon, V. Berisha, V. Atti, and A. Spanias, "Experiments with sensor motes and java-dsp." *IEEE Transactions on Education*, vol. 52, no. 2, pp. 257–262, 2009.