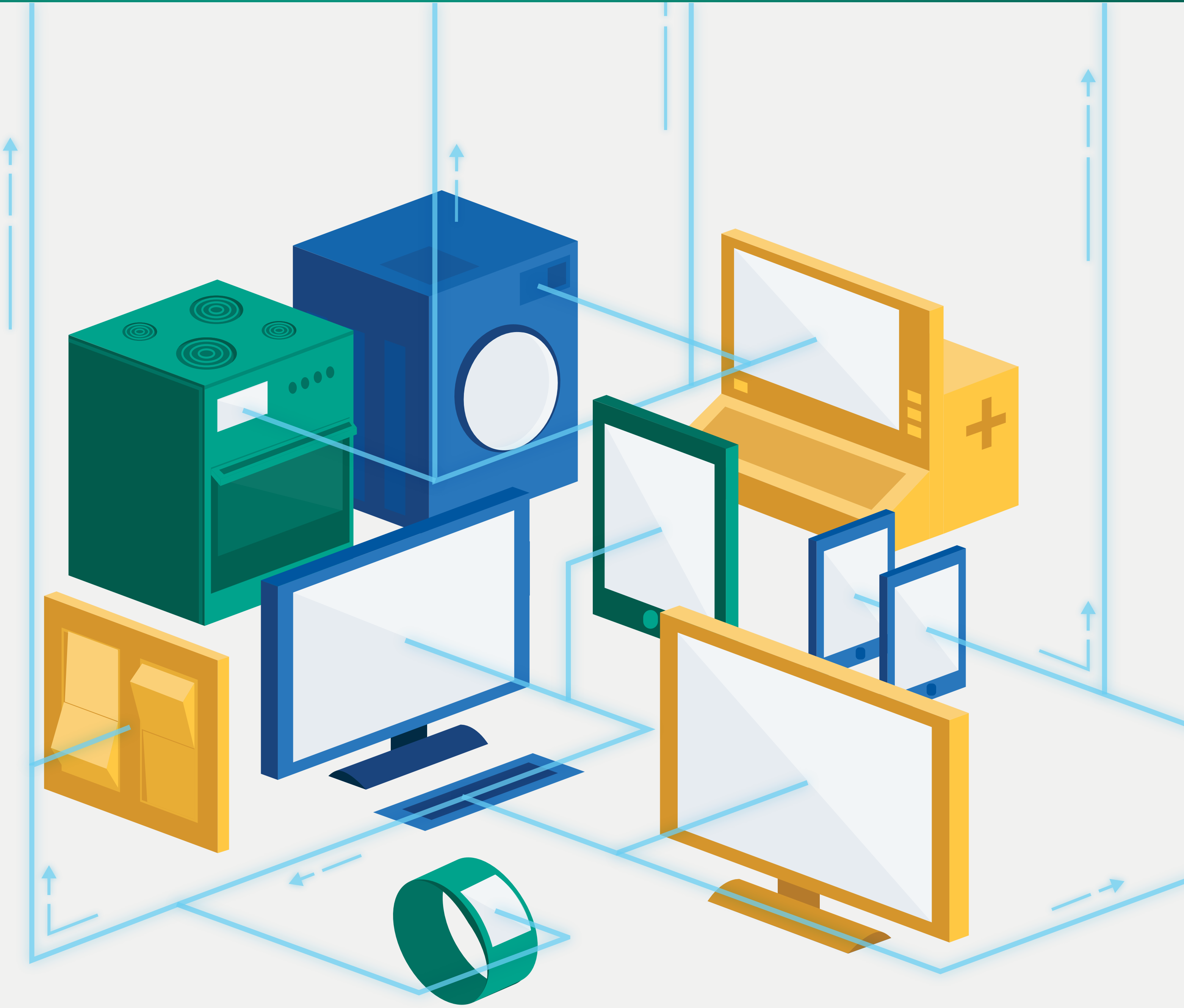
Future-proofing the Connected World:

13 Steps to Developing Secure IoT Products



Presented by the IoT Working Group

CSA cloud security alliance®

Table of Contents

Acknowledgements

Initiative Leads

Priya Kuber
Brian Russell
Dr Shyam Sundaram

CSA IoT Working Group Chairs:

Brian Russell, Leidos

Key Contributors

K S Abhiraj
Sateesh Bolloju
Steve Brukbacher
Giuliana Carullo
Ravi Dhungel
Thomas Donahoe
Raghavender Duddilla
Drew Van Duren
Loïc Falletta
Luciano Ferrari
Ayoub Figuigui
Masaaki Futagi
Mark Grimes
Aaron Guzman
Heath Hendrickson
Sabri Khemissa
Paul Lanois
Jinesh M.K.
Srinivas Naik
Chalani Perera
Sewmi Rajapaksha
Lakmal Rupasinghe
AK Sharma
Mark Szewczul
Srinivas Tatipamula
Sudharma Thikkavarapu
Kim White

CSA Global Staff

John Yeoh
JR Santos

Design

Stephen Lumpe

Letter from the Chair

Much is said lately regarding the need to secure the Internet of Things against a large number of attacks and a diverse pool of attackers. What has been difficult to find however is concise guidance for how exactly to accomplish that goal. One of the primary challenges associated with providing this guidance is the broad applicability of the IoT concept across many industries and many types of products and systems.

The Cloud Security Alliance (CSA) IoT Working Group provided systems-level security guidance in our April 2015 document titled “Security Guidance for Early Adopters of the IoT”. However, an IoT system is only as secure as its weakest link. This document is our attempt at providing actionable and useful guidance for securing the individual products that make up an IoT system - to raise the overall security posture of IoT products.

We hope that this document is found especially useful by those organizations that have begun transforming their existing products into IoT-enabled devices. That is, manufacturers that do not have the background and experience to be aware of the myriad ways that bad guys may try to misuse their newly connected equipment. These manufacturers are often told that there are shortcomings in their security strategy, but have not yet had a good reference guide to help them understand exactly what those shortcomings are and how to fix them.

We also hope that those in the startup communities will find this guide useful. Startups in the connected product/system space are challenged with getting their products to market quickly. Finding the right talent to help secure those products early in the development cycle is not an easy task. This document provides a starting point for creating a security strategy that we hope will help mitigate at least the most pressing threats to both consumer and business IoT products.

It is often heard in our industry that securing IoT products and systems is an insurmountable effort. However, with the help of volunteers like those in our CSA IoT Working Group we can at least attempt to provide a helping hand to product developers that know their products are at risk of compromise but don’t know where to start the process for mitigating that risk. With that I’d like to thank the many volunteers that contributed and reviewed this document and also make mention of two other organizations that are attempting to lend their helping hand with their own IoT guidance- the U.S. Federal Communications Commission (FCC) and the **Securing Smart Cities Initiative**. Both organizations are collaborators with the CSA IoT WG.

Finally, this is a community document and we expect to update this guidance over time. Please direct any comments to our WG and we will incorporate in the next release.

Thanks - and I hope you find this guidance useful.
Brian Russell, Chair IoT Working Group, Cloud Security Alliance

Forward

This is a long document that touches on many points related to securing IoT products in an attempt to be comprehensive. Oftentimes there is a need to quickly identify the critical security items to consider in a product development lifecycle. To aid the reader, we have outlined a Top 5 security considerations list here. It is our opinion that at least focusing on these top 5 items will begin to increase an IoT product's security posture substantially. This list is not a substitute for reading the rest of the guidance included herein, however it does provide a means to get teams started in the right direction quickly.

IoT product developers should start with the following security engineering practices:
1. Design and implement a secure firmware/software update process
2. Secure product interfaces with authentication, integrity protection and encryption
3. Obtain an independent security assessment of your IoT products
4. Secure the companion mobile applications and/or gateways that connect with your IoT products (e.g., encryption/privileges/authentication)
5. Implement a secure root of trust for root chains and private keys on the device

Introduction

This document is meant to provide developers of IoT devices with an understanding of the security threats faced by their products. We also talk about the tools and processes that can be used to help safeguard against those threats. Although IoT systems are complex, encompassing devices, gateways, mobile applications, appliances, web services, datastores, analytics systems and more, this guidance focuses mainly on the 'devices' (e.g., the Things).

ITU-T Y.2060 defines a device in the context of IoT, as a "piece of equipment with the mandatory capabilities of communication and the optional capabilities of sensing, actuation, data capture, data storage and data processing." The concept of a secure IoT device however is not so well defined. For purposes of this document, we define a secure IoT device as a device that implements sufficient security measures such that an attacker will move on to another target. Nothing that is connected is completely secure, however it is possible to make it sufficiently resource-expensive to compromise, that an attacker will deem it illogical to continue down that path.

The composition of an IoT device can vary - It could be a simple sensor that runs a minimal set of firmware on top of specialized hardware and has no need for an Operating System (OS). Or, it could be a stand-alone appliance with minimal of full complexity, requiring a real-time OS and even some software that supports a logic implementation. IoT devices can also be things that implement fully functional web servers that support configuration options on the device. It could also be as advanced as a connected car with hundreds of Electronic Control Units (ECUs), varied wireless communication interfaces and millions of lines of code. **In this document we use the terms IoT device and IoT product interchangeably.**

Document Scope

This secure design and development guidance provides:
1. A discussion on IoT device security challenges.
2. Results from an IoT security survey conducted by the CSA IoT WG.
3. A discussion on security options available for IoT development platforms.
4. A categorization of IoT device types and a review of a few threats.
5. Recommendations for secure device design and development processes.
6. A detailed checklist for security engineers to follow during the development process.
7. A set of appendices that provide examples of IoT products mapped to their relevant threats.

The Need for IoT Security

The IoT is here and is already beginning to transform consumer, business and industrial processes and practices. The year 2015 saw the market adoption of many types of IoT products, we began to see real research that shows the concerns about IoT security are real. Based on the research that we will briefly discuss in this section, we can begin to understand some of the high level needs for IoT product security. These needs include:

- The need to protect consumer privacy and limit exposure of PII and PHI
- The need to protect business data and limit exposure of sensitive information
- The need to safeguard against IoT products being used in DDoS attacks or as launching points into the network
- The need to guard against damage or harm resulting from compromise of cyber-physical systems

We will now provide a discussion on issues and trends within the IoT space. We provide this discussion to point out lessons learned that can be applied by IoT product developers.

IoT Products Can Compromise Privacy

Rapid7 recently published a report on Baby Monitor Exposures and Vulnerabilities. In this report, they described a set of vulnerabilities potentially exploited based on 1) physical access to the device, 2) direct access to the local area network (LAN), and 3) via the Internet.

Other products used within the home have also been compromised. VTech, the manufacturer of high-tech educational toys for children such as electronic learning devices, **announced** that it suffered a security breach in December 2015, exposing personal data of **12 million people**. The interesting take-away from this event was that the devices themselves were not compromised, however the online services that devices connect with were not sufficiently secured. Specifically, a **report** stated that the breach exploited a SQL injection vulnerability and the account registration services did not use TLS. Consider that these vulnerabilities were not related to flaws in the IoT devices themselves, but instead to flaws discovered in the infrastructure that supports the devices. IoT devices do not operate in a vacuum, they are a part of a much larger ecosystem that must also be secured sufficiently.

Wearable products can also introduce privacy concerns. As an example, a study done by **Open Effect** showed that many wearable manufacturers had not yet taken advantage of the new privacy features designed into the Bluetooth 4.2 specification. It was found that tracking a person with a wearable was theoretically made possible when developers relied on publicly discoverable static MAC addresses for their devices.

Lessons Learned

1. Encrypt all acount registration using Transport Layer Security (TLS)
2. Implement software assurance techniques within your development team
3. Thoroughly review protocol specifications for security/privacy updates

IoT products can lend their computing power to launch DDoS Attacks

If you consider the substantial quantities associated with IoT products, you can begin to imagine how useful these products can become for those wishing to perform a Distributed Denial of Service (DDoS) attack. These attacks have already begun. Researchers at Sucuri released a **report** that noted the compromise and use of over 25,000 Closed Caption TeleVision (CCTV) devices towards a DDoS attack. The researchers pointed to a remote code execution (RCE) flaw exposed by some vendors in the market. This showcases again the need for software assurance techniques to be employed by vendors in the IoT market.

Compromising IoT products for use in botnets does not have to be as complex as identifying an unpatched vulnerability and exploiting that vulnerability. Some IoT products ship with no password protections, or use default passwords such as admin for local access. Attackers that identify these low hanging fruit can victimize large populations of the product quickly and employ for their malicious purposes. A real-world example of this is the Lizardstresser DDoS botnet. Security firm Arbor Networks **noted** that the actors running this botnet have begun targeting IoT devices that share default passwords across device classes.

Making things more interesting is the ability to quickly find IoT products that may not have proper authentication in place. A web crawling service ‘Shodan’, crawls the internet at random looking for IP addresses with open ports. If the port lacks authentication, the script takes a snapshot and moves on. This data is searchable publicly.

Lessons Learned

1. Implement software assurance techniques within your development team
2. Never ship IoT products without password protections
3. Do not share default passwords accross a class of devices without requiring immediate password updates on first use

Medical Devices and Medical Standard Protocols are Vulnerable to Attack

Concerns related to the security of connected medical devices are nothing new and have been around even prior to the popularity of the IoT. In 2008, researcher Daniel Halperin and colleagues published the document titled: Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses. This document discussed wirelessly reprogrammable implantable medical devices (IMDs) that include pacemakers, cardioverter defibrillators and implantable drug pumps. The researchers outlined not only privacy issues associated with these devices, but also the ability to change device settings, change or disable therapies, and even deliver a shock to the patient. Read the full report **here**.

Further, researcher Jeremy Richards revealed a slew of **security flaws** in a line of infusion pumps [1]. These included:

- Lack of authentication for telnet sessions;
- Wi-Fi Protected Access Key (WPA) for hospital wi-fi network stored in plain-text on the device;
- The device was running a vulnerable version of a webserver (e.g., the webserver component was not patched); and
- A hard-coded credential was assigned to the File Transfer Protocol (FTP) service.

Vulnerabilities affecting **pacemakers**, **hospital drug pumps** and other medical devices are extremely dangerous and could result in life-threatening injuries or death. Researchers have used **iStan** to demonstrate these vulnerabilities. The US Food and Drug Administration (FDA) has for example issued **an alert** to warn healthcare organizations about the risks associated with the use of certain infusion systems.

Lessons Learned

1. Implement software assurance techniques within your development team
2. Authenticate access to all ports
3. Encrypt keys that are stored on devices
4. Provide an ability for customers to easily keep software components (e.g., web servers on the device patched)
5. Do not share default passwords accross a class of devices without requiring immediate password updates on first use

Drones Are Approaching Mainstream Status and Being Used as a Platform for Reconnaissance

Small Unmanned Aerial Systems (sUAS), also known as drones are have become popular and security researchers are beginning to investigate how to use these systems for reconnaissance activities. This includes the identification of IoT products running certain RF protocols in a geographic area. Researchers at Praetorian showed that drones are an effective tool for this when they mapped ZigBee beacons in 2015.

Lessons Learned

1. Carefully evaluate the chosen IoT communication protocols for your product and configure in modes that limit the amount of information shared

Critical national infrastructure can rely on the IoT ecosystem

From a standpoint of interconnected smart cities or smart factories, we see that a connected smart grid of vulnerable things, a smart network of traffic lights or a smart factory that relies on massive interconnection can have a direct effect on the functioning of a modern society. If we think of a typical Industrial Control System (ICS), we see that there are many areas of concern. These areas of concern include the connection of systems that were never designed to connect to the Internet, the use of legacy protocols that have no security mechanisms built-in, as well as the fact that these cyber-physical systems (CPS) can cause harm and damage if compromised.

Lessons Learned

1. Begin a move toward upgrading legacy protocols to more secure choices within Cyber Physical Systems (CPS)
2. Incorporate Safety Engineering into IoT/ CPS product designs
3. Implement secure interace connectivity within your IoT products

Critical national infrastructure can rely on the IoT ecosystem

Connected Vehicle (CV) technology will enable myriad use cases that support reductions in the numbers of traffic incidents and injuries related to those incidents. Connected Vehicles will communicate using the Dedicated Short Range Communications (DSRC) protocol, which supports rapid transmissions of messages (e.g, Basic Safety Messages - BSMs) between vehicles (V2V), infrastructure (V2I), and applications (V2X). The CV ecosystem consists of a number of technologies, hardware and software components. Infrastructure equipment such as traffic signals broadcast messages such as: 1) Speed limits, 2) Signal Phase and Timing, 3) The presence of traffic conditions ahead. Pedestrians may even use apps on their smartphones or purpose-built dongles to communicate with vehicles and infrastructure equipment.

The security of the entire CV ecosystem comes down not only to defining secure interconnection points, but also implementing secure products to begin with. These products may be the vehicles themselves, roadside units, infotainment systems, or even the 3rd party dongles that we are now starting to see sprout up on the market.

We have already seen research that results in compromise of traffic management platforms as well as vehicles themselves.

Lessons Learned

1. Implement software assurance techniques within your development team
2. Do not share default passwords across device classes without requiring immediate updates to the passwords
3. Implement secure interface connectivety within your IoT products
4. Encorporate Saftey Engineering into IoT/CPS product designs
5. Implement secure interface connectivity within your IoT products

Moving Forward

Whether we're dealing with smart farms or connected manufacturing facilities, sensors and robotics are now beginning to work hand in hand towards the accomplishment of objectives. New research in the area of blockchain technology shows promise in extending these capabilities through machine-to-machine negotiation. These capabilities are driving the human out of the decision making loop in many instances and as we rely on IoT products to do the basic thinking for us, we will need to make sure that those products and their associated services and interconnection points are each developed as securely as possible.

Why Development Organizations Should Care About Securing IoT Products

With predictions of 50+ billion devices connected by 2020, IoT products will be widely deployed and gain entrance into our homes, workplaces, vehicles and even airplanes. Adding interconnectivity between these devices and our existing network infrastructures will open up new attack vectors that many will attempt to exploit. Security researchers today are working hard to identify vulnerabilities associated with many of the existing IoT products however not all vulnerabilities will be identified and patched prior to a malicious actor making use of them. The consequences of a particular IoT product being used to compromise sensitive user information or worse, to cause harm or damage, will be catastrophic to the product vendor.

Even knowing the severe ramifications of fielding an IoT product used in a large scale attack, security professionals are often faced with a daunting task of providing a business case for expending funds to secure products and systems. Because of this, we have enumerated some reasons that developers should care about securing IoT products. According to [24] security must be addressed throughout the device lifecycle, from the initial design to the operational environment which includes Secure booting, Access control, Device authentication, Firewalling and IPS, finally Updates and patches. Providing a sufficient budget for IoT secure product engineering:

1. Reduces the likelihood that an IoT product will be counterfeited
2. Limits the ability for attackers to compromise your customer's privacy
3. May limit liability in the case of a compromised device
4. In the case of CPS, may limit the ability for an attacker to cause damage or harm
5. Reduces the likelihood that your product will receive negative press
6. Can reduce the potential risk and have mitigation controls in place in case the product does have any limitations to embed the security into it

In the case of a compromised IoT product, the resultant impact to your business is almost never positive. At a minimum, customers will hesitate to purchase the product based on knowledge that it leaves their systems or homes vulnerabile. At worst, a compromised IoT product will invite legal action or cause physical damage or harm.

The US Federal Trade Commission (FTC) has even laid down certain **IOT guidelines** advising the developers to build security into the architecture, rather than as an afterthought. Other factors that contribute to consumer IoT device vulnerabilities are a mixture of bad design, non-regulated or insufficiently regulated environments, the use of third party hardware/ software and insecure cloud-based resources. Since there is a lot of variability and interdependence as mentioned, it is easy to pass the blame to any of these services as the number of interdependencies increase (overall multiple points of blame).

IoT Device Security Challenges

In order to understand how to design and develop IoT products securely, it is important to first understand what challenges security engineers face when deciding on a security approach. Here we define a set of these challenges:

Table 1

Criticality	Challenge
HIGH	IoT products may be deployed in insecure or physically exposed environments
HIGH	Security is new to many manufacturers and there is limited security planning in development methodologies
HIGH	Security is not a business driver and there is limited security sponsorship and management support in development of IoT products
HIGH	There is a lack of defined standards and reference architecture for secure IoT development
HIGH	There are difficulties recruiting and retaining requisite security skills for IoT development teams including architects, secure software engineers, hardware security engineers, and security testing staff
MEDIUM	The low price point increases the potential adversary pool
MEDIUM	Resource constraints in embedded systems limit security options

IoT products may be deployed in insecure or physically exposed environments

IoT products that are left physically exposed are vulnerable to being stolen and reverse engineered to identify secrets (e.g., shared keys/passwords, etc) or identify vulnerabilities in the software that can then be exploited at a later time on operational products. Consumer IoT products often operate within environments (e.g., homes) that typically have limited to no security controls in place to protect against the threat of attackers exploiting vulnerabilities in software. There is limited to no concept of defense-in-depth applied in this scenario and IoT product developers should anticipate the product operating within a network where:

- WiFi routers may be setup to use weaker authentication mechanisms
- WiFi routers may be old and unpatched, leaving them exposed
- Home computers may run older versions of operating systems, no virus detection and be unpatched
- Tablets and smartphones attached to the network may be running outdated firmware/software
- Gaming consoles may be attached to the network

Attacks against insecure WiFi routers, smart phone applications and even insecure peer IoT products within the home can expose keys used to establish network connectivity or even trust relationships.

Operating Securely** in a **Challenging Environment

1. Apply policy based security to force IoT products to update latest security critical fw/sw
2. Identify flexible self-service identity management capabilities for IoT products
3. Encrypt indentify/key material within mobile applications when used to establish trust relationships with IoT products

Security is new to many manufacturers and there is limited security planning in development methodologies

Traditional IT security is a specialized discipline that those in the computer industry have taken many years to embrace. For IoT product developers that have never had to deal with security concerns, it is understandable that they would often lack the skills required to engineer and develop products in a secure manner. Many product developers that are adding software and connectivity to their portfolios have begun to build the skill set necessary to understand the security weaknesses inherent in connected devices and the need to understand the basics of software assurance. There is still a gap however as some products have long-lead product cycles which will lead to a continued influx of insecure devices onto the market.

Security evaluation and certification provide IoT product developers with a competitive advantage over their peers and increase the viability of product deployment in many environments. Regulatory, privacy and compliance mandates require that devices added to certain systems be tested and verified prior to deploying especially when dealing with sensitive information (e.g., PII, etc). For developers that continue to lack the security expertise needed to begin securely developing their products, there are many 3rd party organizations that offer security evaluation services for a fee or even at no charge.

Operating Securely** in a **Challenging Environment

1. Create an IoT security training program for the development team
2. Indentify and participate in threat sharing (e.g., ISAC) initiatives and establish a framework for threat modeling the product
3. Obtain buy-in from senior management on the need to incorporate security into the product

Something that may not seem apparent at first, is that organizations need to embed security into the engineering processes used for product design, development, test and integration. Many developers follow agile principles when developing their IoT products. There are often gaps in agile processes that have to be filled. For example, ensuring a process whereby security requirements are identified well in advance. These security requirements should also be tracked through the entire process to make sure that they are satisfied during the testing of products. The concept of ***secure by design*** and ***privacy by design*** should be considered by IoT product developers.

Organizations developing consumer or enterprise IoT products should follow best development security practices (e.g., Microsoft SDLC, BSIMM, OpenSAMM) which include peer code reviews and code analysis. Penetration tests against the final products in a representative environment should also be conducted. Although software development engineering practices have came long way , the challenge to integrate development, operations and security has been challenging task due to inherent engineering gene of each team. The concept of DevOps and DevSecOps continues to evolve to address deficiencies in engineering methodologies. Penetration Tests inside the organization ensure that software defects are minimized.

Operating Securely** in a **Challenging Environment

1. Review and update your development processes to incorporate security at all stages
2. Incorporate privacy by design principles into all IoT product developments

Security is not a business driver and there is limited security sponsorship and management support in development of IoT products

The CSA conducted a survey of technology startups in 2015 to better understand their motivations related to security of IoT developments. Results from the survey showed that investors and technology startups are not concerned with the security of their products. They are instead focused on getting their products to market quickly and ensuring that core functionality works as expected.

Making things more difficult from an information security perspective, is the need for device developers to consider usability vs. security trade-off. Since many of these IoT devices depend on other devices to function (e.g., a smartphone), the need for security to be planned at the architecture level is higher than ever before. In some cases, manufacturers of consumer-based IoT devices have made conscious decisions to forgo security best practices in an effort to make these devices easier to configure by the homeowner.

Indeed, security is often viewed as a consumer inconvenience and/or complication that is perceived to drive up support costs, diminish user friendliness, etc. Password usage is an example of the trade-offs that can occur between security and usability - the simpler and longer lived the password, the easier it is for the user to remember (and for the adversary to guess). The challenge is for future technical solutions to achieve both increased security and a good user experience whenever possible.

IoT product developers should consider security to be a part of business requirements. Each product being developed should first be examined to understand the unique threats to the product, and then a backlog of security requirements generated to aid in mitigating the potential realization of those threats.

Operating Securely** in a **Challenging Environment

1. Begin each product development with a threat model
2. Derive security requirements from the output of the threat model and track those requirements through to closure

There is a lack of defined standards and reference architecture for secure IoT development

Technologies implemented to support the Internet of Things are growing up in number and span across several fields - e.g., platforms, communication, routing protocols, security features and so on - in order to match the need of a smarter and fully connected ecosystem. [22] suggests a systematic security architecture called IPM which comprises three essential security perspectives which are information, physical and management model. PubNub [23] is another good example which is a secure global Data Stream Network and an easy to use API that enables customers to connect, scale, and manage IoT devices and realtime apps. Beside them, both industry and academia are trying to move in the direction of a standardized IoT environment. To this end, indeed, tons of standards have been proposed. The Internet Engineering Task Force (IETF) is currently leading the standardization process in terms of communication protocols for resource constrained devices, developing - at the same time - several Internet protocols, including the Routing Protocol for Low Power and Lossy Networks (RPL) and Constrained Application Protocol (CoAP). Many other communication and messaging protocol standards have been proposed, together with authentication and authorization standards. Anyway, understanding which to choose and how best to apply them to unique product developments is non trivial.

One of the others main concerns in terms of standardization is that there is no accepted reference architecture among vendors, and this is important since IoT products and services require the cooperation of many technologies and protocols. Indeed, even if a reference architecture called **IoT-A** has been proposed during a funded European project, it has not been accepted as standard. To mitigate this, many IoT product developers choose an IoT platform as a starting point, and start to build up their customizations and services from there. However those platforms themselves often are not interoperable with each other. Further, developers and designers move independently and may often not make secure choices. There are also gaps in standards themselves, such as a lack of an agreed-upon method for offloading security log files from constrained IoT devices.

Operating Securely** in a* ***Challenging Environment

1. Carefully evaluate the environment in which devices are deployed, and choose technologies accordingly to the required security level
2. Evaluate the performance vs security tradeoff, exploiting the best matching protocol stack in order to reduce security risks and breaches
3. Evaluate the security features offered by the IoT components (e.g., TPM hardware, etc) and use whenever possible

There are difficulties recruiting and retaining requisite skills for IoT development teams including architects, secure software engineers, hardware security engineers, and security testing staff

IT security staff are consistently challenged with learning new technologies and products, however the IoT introduces even more challenges to keeping staff sufficiently trained. Product Security Officers and their teams now have to concern themselves with vulnerabilities within software (to include new languages that have not typically been used in the past), ways that attackers can compromise their product's hardware features, and of course secure mechanisms for creating and distributing firmware and software updates to thousands if not millions of devices.

New training must provide security teams with an understanding of:

1. The new technologies associated with the IoT
2. New threat profiles associated with IoT vulnerabilities
3. The impact of a compromise of IoT systems (across potentially millions of devices)

Operating Securely** in a* ***Challenging Environment

1. Create an IoT security training program for the development team

The low price point increases the potential adversary pool

As discussed in other parts of this paper, the low cost of typical IoT products, especially consumer devices, makes it simple for both researchers and malicious actors to acquire and spend time finding security issues and analyzing the security protections built into each device. This allows for the systematic discovery of security vulnerabilities related to both the hardware and software, knowledge of which can then be used to exploit weaknesses in operational environments.

In situations where an attacker has been able to find and reverse engineer an IoT device, critical data such as shared symmetric keys, private keys and other cryptographic primitives may be compromised. In the case of an IoT network that relies on shared symmetric keys, this is significant since the attacker would likely have access to all data held within other devices that share the same key. Identifying that default passwords are used on devices would also lead to the ability to easily compromise like-devices within a network.

Operating Securely** in a **Challenging Environment

1. Consider physical safeguards such as tamper detection to guard against physical access to sensitive internals
2. Lock-down physical ports (including test ports) on the product using passwords

Resource constraints in embedded systems limit security options

Although a few years back there were significant concerns related to IoT products being able to afford (e.g., size, weight, power) sufficient security mechanisms, those concerns are not as significant today. Of course, there are still some highly constrained devices that will not be able to implement a cryptographic hardware module, however progress related to the creation of software based cryptographic modules and even the ability to add in things like security co-processors to MCUs makes this challenge less pressing.

For now however, all IoT devices will not support robust security mechanisms. In many devices, there will be tradeoffs made between size, weight power and memory / processing power. As technology continues to improve these constraints may be eased. As IoT product developers today, be cautious in your choice of technologies that underlie your IoT product.

Operating Securely** in a **Challenging Environment

1. When possible, use hardware-based security controls to safeguard sensitive information

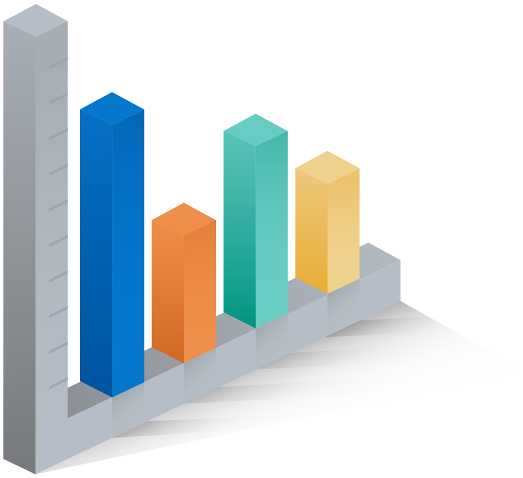
IoT Security Survey

The CSA IoT WG conducted a survey of startup organizations developing IoT solutions. The results and analysis below demonstrate the need for this secure development guidance document. Organizations were requested to provide answers related to their secure development methodologies and practices. Key findings from the survey include:

1. Startups don't consider information stored on a device as sensitive (any sensitive data is stored on a server).
2. Users want to share information (sharing mentality).
3. Startups rely heavily on the use of COTS services.
4. Most startups are using AES, although most also consider encryption to be not important.
5. Most devices don't share a master key shared across devices; admin at server side.
6. No security applied to the development environment.
7. No threat modeling of products.
8. No secure firmware updates.
9. Investors don't seem to care about security, much more focus on functionality.

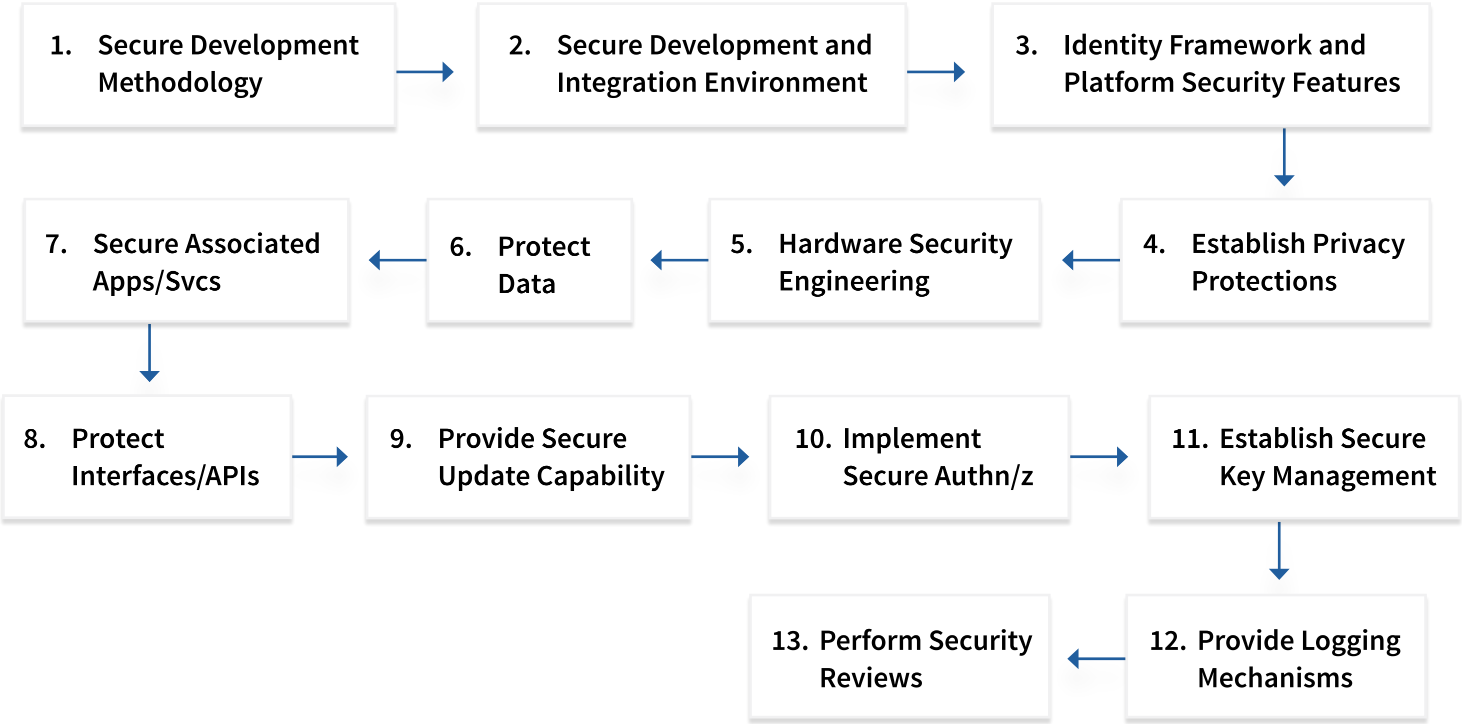
Based on our survey results, it is easy to understand that security is not a foremost thought in the minds of many IoT developers. Given the need to get IoT solutions to market quickly, this is not necessarily surprising, however it does cause concerns related to the posture of new devices that will hit the market over the coming years.

Another interesting aspect of the survey results was that there is a heavy emphasis on the sharing mentality. That is, information should be shared and it is more important to enable sharing of information versus securing it. While this is often true for many commercial IoT devices, it is often not true for businesses that place an equal weight on securing and sharing information. In addition, of particular concern is the potential introduction of these consumer devices into the enterprise as employees and/or contractors brings their devices into the corporate environment either with or without permission.



Guidance for Secure IoT Development

This section provides considerations and guidance for designing and developing reasonably secure IoT devices. Note that this guidance is not meant as a substitute for understanding fundamental system security engineering methodologies and techniques, but instead aims to mitigate some of the more common issues that can be found with IoT device development. We outline a number of activities that will enable a development organization to begin enhancing the security posture of IoT products. Figure 2 provides a graphical view of the steps toward developing more secure IoT devices.


1. Secure Development Methodology
2. Secure Development and Integration Environment
3. Identity Framework and Platform Security Features
4. Establish Privacy Protections
5. Hardware Security Engineering
6. Protect Data
7. Secure Associated Apps/Svcs
8. Protect Interfaces/APIs
9. Provide Secure Update Capability
10. Implement Secure Authn/z
11. Establish Secure Key Management
12. Provide Logging Mechanisms
13. Perform Security Reviews

1. Start with a Secure Development Methodology

Today most software development teams do not focus sufficiently on security in the requirements, architecture, design and code base. If that is the case then it is a lot to ask that product companies interested in “connecting” their existing products have an expertise in how to do it securely. In order to tackle this challenge, we need to fall back on a secure engineering approach - a.k.a a secure development methodology.

Ideally, an organization’s secure development methodology will call out the need for more than just technological checks. Things like documentation, peer reviews, and incorporating security requirements into the product lifecycle should be incorporated as well. The engineering process should also include substantial **feedback loops**, aimed at making the product better and more secure.

Examples of these feedback loops include the addition of product backlog items upon identification of a vulnerability in code, or the update of product design approach upon identification of issues within integration testing (e.g., Continuous Integration tests).

Perform Threat Modeling

Threat modeling is a core component of a secure development methodology. The emergence of IoT technologies and products is a constantly changing landscape. It is hence important to ensure the reference to a set of threats and issues to help ensure they are addressed appropriately. One such good reference is the Open Web Application Security Project (**OWASP**), which defines the top ten security surface areas presented by IoT systems. OWASP provides information on aspects such as threat agents, attack vectors, vulnerabilities, and impacts associated with each. This applies to manufacturers, developers, and consumers to help better understand the security issues associated with the IoT, enabling users in any context to make better security decisions (building, deploying, or assessing) related to IoT technologies. The **following** are some broad areas that are covered:

- The Internet of Things Device
- The Cloud
- The Mobile Application
- The Network Interfaces
- The Software
- Use of Encryption
- Use of Authentication
- Physical Security
- USB ports

As part of the secure development process Threat modelling must be conducted on the software or hardware to identify the potential threats and appropriate mitigation controls must be put in place to mitigate the identified threats. General software threat modelling techniques would still apply for IoT.

References to Threat Models:
Microsoft Threat Modeling
OWASP Application Threat Modeling
IEEE Cybersecurity Secure Design

A reference for threat modeling can also be found in Adam Shostack’s **book** “Threat Modeling: Designing for Security.” Microsoft also defines a well-thought-out threat modeling approach using multiple steps to determine the severity of threats introduced by a new system. The threat Modeling approach (based on Microsoft SDL):

It is also good to examine the different types of threats to IoT devices.
Table 2 provides a view into the categorization of threats.

Table 2

Threat Type	Discussion
Spoofing Identity	Examine the system for threats related to the spoofing of machine identity and the ability for an attacker to overcome automated trust relationships between products, devices and services. Carefully examine the authentication protocols employed to set up secure communications between various devices (M2M) and between devices and applications that make use of data provided by these devices. Examine the process for provisioning of identities to each IoT device (bootstrap processes). Do not share identities across devices in a family.
Tampering with Data	Identify configuration restrictions (r,w,x) that must be placed on files held on the IoT product. Implement the concept of least privilege. Ensure that smartphone applications that are paired with the device are granted only the permissions that are absolutely required to perform a function.
Repudiation	Incorporate health checks to verify the security functionality of the device prior to operational use. Incorporate authentication and integrity protections on data being generated by the IoT product. Design in methods to restrict the replay of messages (e.g., sequence numbers / timestamps). Validate the integrity and authentication controls placed on cloud data stores as well as mobile app data stores.
Information Disclosure	Restrict disclosure of information via APIs to other IoT products to only that which is necessary. Validate that encryption is employed on smartphone apps and cloud services to restrict access to information provided by the IoT product. Implement data-at-rest encryption for any IoT products that store/retain information deemed sensitive, and store cryptographic keys associated with this operation is a hardware security container when possible. Implement techniques such as key zeroization for IoT products that store or process sensitive information. Use a cryptographic protocol to encrypt any communications with peer devices, smartphone applications and cloud services (e.g., TLS/ DTLS).
Denial of Service	Guard against denial of service attacks targeting your IoT product's cloud services when devices are directly connected to the cloud. Implement rate-limiting on APIs as needed. Guard against Iot devices being taken over within botnets by providing customers with efficient methods for keeping software components updated and quickly patching any identified vulnerabilities. Be open to receiving inputs from the security community on new vulnerabilities and work quickly with researchers to close those vulnerabilities and make patches available.
Elevation of Privilege	Design based on principle of least privilege for all user and service roles. Limit privileges associated with smartphone applications and the IoT product to only that which is required, especially when integrating with partner IoT products.
Bypassing Physical Security	Consider your threat environment. In certain instances, robust physical security may be required. At a minimum, debug ports should be password protected and disabled if allowable. Techniques such as tamper detection and response are advisable for IoT products designed to support critical infrastructure use cases.

Appendix A provides a detailed examination of categories of IoT devices and their typical threat exposure. We suggest that you use this information as a reference when designing your IoT device security features.

Security Requirements

The concept of defining security requirements for your product should also be explored and implemented. Within Government programs, requirements culled from DISA Security Technical Implementation Guides (STIGs) or Security Recommendation Guides (SRGs) are used to populate the product backlog. This allows for traceability of the requirement through the development process and ensures that at least a minimal set of security controls have been implemented within the device.

A secure development methodology should focus on more than just development however. Developers should take responsibility for building secure processes into their products as well. To understand what this entails, we should look to the Building Security In Maturity Model (BSIMM) from security firm Cigital.

Security Processes

BSIMM allows organizations to compare existing development best practices to those used in many different organizations. BSIMM consist of four domains and 12 practices. The applicability of each of these practices to the IoT is discussed in Table 3.

Table 3

BSIMM Doman	Practice	IoT Applicability
Governance	1. Strategy and Metrics 2. Compliance and Policy 3. Training	• Establish a strategy and plan for measuring your security controls against industry benchmarks • Build relationships with 3rd party test and consulting organizations. • Identify regulations and laws that the device must comply with in an operational environment and track compliance • Train all team members on roles related to securing IoT devices and on typical attack patterns for exposed IoT devices
Intelligence	4. Attack Models 5. Security Features and Design 6. Standards and Requirements	• Research and perform trade-offs to determine optimal security features • Examine ways that similar products have been compromised and take into account lessons learned • Build attack models, patterns and reference architecture etc.
SSDL Touchpoints	7. Architecture Analysis 8. Code Review 9. Security Testing	• Design layered security features that incorporate both electronic and physical security as needed and support confidentiality, integrity and availability of IoT product services • Mitigate and Reduce the attack surface of the IoT device as much as feasible • Design associated services to scale to large quantities of devices, securely • Evaluate the software and hardware during security testing

Table 3

Deployment	10. Penetration Testing 11. Software Environment 12. Configuration Management/ Vulnerability Management (CMVM)	• Perform penetration testing in an operationally relevant environment to identify potential weaknesses • Establish processes for effective configuration management of the IoT device (during deployment)

Perform Safety Impact Assessment

A unique differentiator of the IoT is the blending of physical and electronic worlds. This means that breaking into an IoT device or service can cause a physical reaction. With larger IoT devices (e.g., connected cars), it is obvious to see the potential impact caused by a malicious event. Even smaller device compromises can result in harm or damage however (e.g, smart home door locks, etc). Developers should take this into account and perform a Safety Impact Assessment as part of their design process.

Consider the safety impacts of a product or service compromise. Given the intended usage of the product, is there anything harmful that could happen if the device stopped working altogether (e.g., denial of service)? If the device by itself is not safety critical, are there any other devices or services that are safety critical and depend upon it? How could potential harm (from device failure) be minimized or avoided? What issues might others consider safety-related or harmful? Are there any other similar or related deployments that have been considered safety relevant or have done harm?

2. Implement a Secure Development and Integration Environment

The need to develop software and hardware systems securely is not new. There are industries that have tackled the challenges of how to do so already and we should try to learn from their efforts to help develop IoT products securely. A great example of this is the Motor Industry Software Reliability Association (MISRA) secure coding efforts for C and C++. MISRA is designed for safety-critical systems and incorporates many of the items we discuss in this paper, including: software development process, training, coding styles, tool selection, test methodology and verification procedures (**reference link**). One of the primary goals of MISRA is to avoid undefined results as these potentially become critical issues when dealing with products in the Cyber Physical Systems (CPS) category. This should be a goal of all IoT product developers though.

With this in mind, we now review a section of programming languages that are popular for use with IoT products, and provide pointers to detailed security guidance (when available) for each of these languages.

Evaluate Programming Languages

Languages often used today include JavaScript along with Node.js, as well as languages traditionally used for embedded systems (C). Java, Python and various other languages are all also well represented as IoT programming languages.

Remember also that we are not only talking about developing secure IoT devices. Developers of IoT products are very often also responsible for developing the smartphone applications that interact with the device and the cloud services that collect information from the devices. Security guidance for the appropriate programming language should be reviewed in that regard as well.

Information Week **published a summary** of the top programming languages used by IoT developers. Table 3 provides a view of those and other programming languages that might be selected to develop IoT products. IoT product developers should familiarize themselves with the appropriate security guidance.

Table 4

Language	Recommended Security Guidance
C	• Guidelines for the Use of the C Language in Critical Systems, ISBN 978-1-906400-10-1 (paperback), ISBN 978-1-906400-11-8 (PDF), March 2013. Motor Industry Software Reliability Association (MISRA) • SEI CERT C Coding Standard **Link** • Motor Industry Software Reliability Association (MISRA) **Link**
C#	• Security (C# Programming Guide) **Link** • OWASP **Link Link** • CodeGuru **Link**

Table 4

C++	• Guidelines for the Use of the C++ Language in Critical Systems, ISBN 978-906400-03-3 (paperback), ISBN 978-906400-04-0 (PDF), June 2008. • Motor Industry Software Reliability Association (MISRA) **Link** • SEI CERT C++ Coding Standard **Link**
Embeded C++	• **Dialect of the C++ programming language for embedded systems**
Erlang	• **Link** • Rikitake, K and Nakao, K. Application Security of Erlang Concurrent System. Network Security Incident Response Group, NICT, Japan.
Objective C	• Secure Coding Guide **Link**
Java	• Oracle Secure Coding Guidelines **Link** • The CERT Oracle Secure Coding Standard for Java **Link** • Android Secure Coding Guidelines **Link**
JavaScript	• OWASP AJAX Security Guidelines **Link**
Python	• OWASP Python Security Project **Link**
Rust, Go, ParaSail, B#	• Parallel Specification and Implementation Language **ParaSail** • **Go**, **Rust**, **B#** (object-oriented, and multi-threaded programming language embedded systems)

Integrated Development Environments

Developers will typically need to identify a MCU and a framework to use as the starting point for the product, as well as any sensors that are required for device operation. Identification of the security services to make use of from the MCU is a critical aspect of selection here, although there are add-on hardware components that can be integrated in cases where the MCU lacks the ability to implement security functionality on its own. All of these hardware components are assembled and tested while developers work on the firmware/software features of the product.

In some cases, robust software capabilities are required. As an example, an IoT product may be configured with a lightweight web server that supports user-based configurations through a web browser on a connected computer or mobile device. Also, a real-time operating system may be selected that provides support for various protocols. Often, third party applications are integrated into the device and in most cases APIs are employed for connectivity to upstream services and other devices.

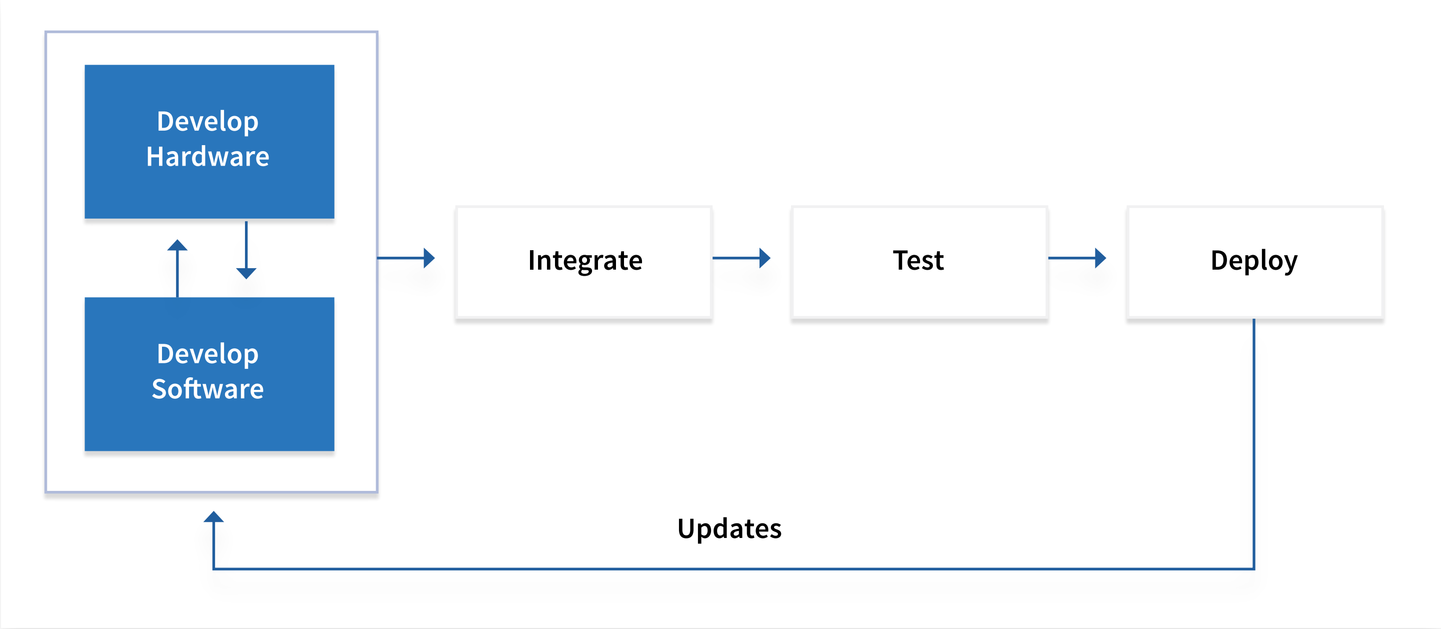
IoT products may be developed in a stand-alone fashion, but often it makes sense to leverage development kits. Many kits are available now that provide interoperability capabilities across device types or come out of the box with connectivity to the cloud.

All of these technologies are brought together during development, and in most instances additional software is written that provides the differentiating features of the product. As can be seen, the need to employ a methodical process for the development, integration, test and deployment of these products is required. Security should be a consideration during each of these stages of the development cycle.

Integrated Development Environments (IDEs) are often used by software developers. The IDEs themselves can offer some set of security services. Although there are many IDEs available for use, examining the Eclipse IDE specifically we can see that there is Interactive Static Analysis support to developers (for Java and PHP) to detect and mitigate software vulnerabilities in the code through the **OWSAP ASIDE project**. Another plugin is **Contrast for Eclipse** which covers the following:

1. Automated detection of OWASP Top 10 vulnerabilities
2. Transparent integration with the Eclipse IDE
3. Detailed data flow analysis of the running application
4. Vulnerability traceback in the source code (5) Context sensitive expert security advice. There is also an **IOT eclipse** project that helps to build IoT solutions around open source projects.

Figure 1



Continuous Integration Plugins

CI servers such as Jenkins, offer flexibility with well known plugins and testing tools such as **OWASP ZAP** (**Jenkins ZAP Plugin**) to be added into an enterprise SDLC. ZAP allows for code to be dynamically automatically scanned after build steps are complete and ready for acceptance tests. Pending ZAP's automated assessment report, a build can be deployed if the scan has been found to pass or return back to the design stage if the scan results in a failure. A typical scan should check for unauthorized file access, outdated application infrastructure, various parameter injection attacks and a number of functional security features such as Login and Logout forms. There are many testing **frameworks**, scanning policies, and sequence configurations available when using ZAP that can be used for better integration into the software development lifecycle. It is always recommended to support automated scanning with manual testing to ensure security vulnerabilities have been accounted for.

Note that ZAP alone does not provide full detection of IoT related vulnerabilities. Researchers have found issues with vulnerabilities related to memory management, hardcoded credentials, authorization bypass, etc which still must be found and remedied prior to shipment of the IoT product.

Testing and Code Quality

The most important part of developing using an Agile approach is the feedback loop between making and observing change. **Martin Fowler's Test Pyramid** is an ideal approach for testing a product. The product is largely composed of lots of small unit tests that validate each independent component of the system in various ways. Developers can maintain high confidence in a product through fast feedback of a system that does not require the clock time of resetting hw components. Developers can iterate quickly, and integration can be proved through validating the work against the hardware; the stubs are substituted with the hardware interface for a longer-running build & deployment.

It is important to note that the technology and platform often dictate the types and availability of testing tools. Choice of security testing tools is relative to the architecture the product lives in. The same wireless hacking tools used for general purpose hacking can be applied to subsets of IoT architecture. The attack vectors on IoT remain consistent with the technology domain its lives in: a web application is a web application, protocols over supported link layers can be tampered with (packet injection). While there are formal protocols in the IoT domain (**CoAP**, **ETSI SmartM2M**, **MQTT** or **LwM2M**), there have also been formal protocols for SCADA (Supervisory Control and Data Acquisition) (**MODBUS**, **DNP**, **UCA**). The specifications of these protocols address the sensitive areas of the protocol that without proper authentication and access control could be openly fuzzed for weakness. Some security testing tools and often, government compliance criteria (e.g., **STIGS**, NIST 800-53v4) may influence Security in the SDLC due to C&A compliance.

Edge cases must also be factored in, for example where device power and broadband spectrum use is important. And, some testing scenarios may not be possible without government waivers. For example, increasing TX power of wireless card from 20 to 30db by hacking region codes -- Yet these scenarios represent plausible **MiTM** attack scenarios. For example, overpowering another BSSID (basic service set identifier) with the same ESSID (extended basic service set identifier) so that end devices are no longer connecting to the intended access point.

Another important part of the secure development environment is the need for code quality dashboards. An IoT device should be backed with reports that provide information about software quality. These reports are tied to the CI builds and provide unit test results with code coverage metrics and the results of all the static analysis checks.

Processes

Libraries that will be used by IoT products may be open source. Having a process in place to vett the libraries that developers will be using in the products is a good practice. The reviews should at a minimum look to ensure that the libraries being used are from reputable organizations. Additionally, making sure that the legitimate libraries have been downloaded for use should be a concern. Having a secured repository for validated code will allow developers to have confidence that they are not using software that has been tampered with.

Configuration Management (CM) is another area that should receive attention. Tools like **GitHub** and **GitLab** are commonly used today. Monitor source code regularly to defend against the insertion of unauthorized code into your repositories (e.g., **Link**). Also, practice care in ensuring that developer logins are secure and do not upload credentials to the repository. See an informative article on this topic **here**.

3. Identify Framework and Platform Security Features

As you begin to design the technology architecture of your device, you will be faced deciding whether to build a stand-alone IoT device or make use of one of the frameworks that are available. Frameworks offer the ability to develop interoperable devices.

Selecting an Integration Framework

This section examines the security characteristics of some of the popular integration frameworks available today for your IoT development effort. Integration frameworks are especially useful when developing IoT devices that must communicate with peer IoT devices in order to participate in automated workflows. Consider examples within a home environment where devices work together and change state based on inputs from other devices/sensors. When a smart doorbell rings at night it sends a signal to automatically turn on certain lights for instance, or when someone wakes up a smartwatch sends a signal that alerts the coffee maker to begin brewing.

With these peer-to-peer use cases the challenges associated with making them all work securely together become apparent. If these were all manufactured by different vendors, making them work together with disparate protocols, cryptographic algorithms and even certificate formats would be problematic. Integration frameworks provide the tools needed for devices to operate as a single ecosystem, often even without the need for devices to be manufactured by the same vendor.

From a security perspective, you want to make sure that you select a framework that provides your developers with the tools needed to implement this interoperability securely. This begins with the concept of secure onboarding and extends to the ability to support proactive management of the device and secure communications. Let's examine some of the tools that should be evaluated when selecting a framework.

Device Onboarding: This is the process for joining the device to a particular network. Frameworks should provide the mechanisms that enable this process. This includes provisioning of an initial set of network credentials. This should also allow for change of ownership at a later time.

Configuration: This includes the ability to name/rename an IoT device, set passwords, or reset the device.

Asset Management: This includes maintaining knowledge of the location of a device, the hardware/ software profile of a device as well as the ability to maintain firmware/software updates for the device.

Discovery: This is the process for enabling the discovery of other devices and services.

Secure Connections: This includes the protocols and algorithms required to support authentication of the device to other devices/ services; verification of server/ peer device authentication credentials, and encryption of communications. This may include the instantiation of protocols such as TLS and DTLS. Sufficiently-strong cryptographic algorithms should also be made available.

Cloud Gateways: Gateways do not always need to be physically separate devices although they can take that form. These gateways provide a link between a local network and the cloud to support global interactions. Gateways should implement secure APIs to the CSP with encryption, authentication and integrity applied.

Table 5

Framework	Transports	Languages Supported	Platforms	On-Boarding	Asset Mgmt	Config Mgmt	Secure Connection
AllJoyn	WiFi Ethernet Serial PLC	C C++ Obj-C Java	Arduino Linux Android iOS Windows MAC	●	●	●	●
HomeKit	WiFi Bluetooth-LE (on top of HomeKit Accessory Protocol (HAP)	Objective-C	iOS				●
IoTivity		C C++ Java JavaScript		●		●	●
ThingWorx		Java JavaScript					●
Xively		JavaScript	Android iOS	●	●	●	●
Oracle Java Embedded		Java JavaScript					●

Now, let's look at some of the framework providers in more depth. Table 5 provides a description of the security functionality of each of these frameworks.

Table 6

F/W	Discussion
AllJoyn	AllJoyn from the AllSeen Alliance is a collaborative open-source software framework that supports interoperable discovery and communications between IoT devices regardless of the type of device or the operating system used. AllJoyn supports the core capabilities of onboarding, asset management, configuration management and secure connectivity. There are a number of APIs available for each of these **base services**. Policies play an important role in AllJoyn-based devices. Developers can define policy templates that constrain the choices available to device administrators during operation. Policy templates contain a list of permission rules. The template is queried and then used by the security manager application to prompt the user for specific policies. During operation, the administrator role is allowed to query, install, update or remove a policy file. AllJoyn leverages certificates for a number of functions. There are two primary certificate types: **Identity:** Used to assert the identity of the device; **Membership:** Used to assert membership in a group. The security

Table 6

	manager application is used to generate and distribute membership certificates (**reference link**). Encryption is also supported within AllJoyn. Peer-to-peer encryption is enabled using the AES algorithm with a 128-bit key length and authentication can be enabled using either Pre-Shared Keys (PSK) or the Elliptic Curve Digital Signature Algorithm (ECDSA). FIWARE is an open platform born from an European project. It provides a set of APIs and functionalities to enable easy development of Smart Applications in multiple vertical sectors. FI-WARE supports different communication protocols, including MQTT, COAP and OMA LWM2M.
FIWARE	FIWARE is an open platform born from an European project. It provides a set of APIs and functionalities to enable easy development of Smart Applications in multiple vertical sectors. FI-WARE supports different communication protocols, including MQTT, COAP and OMA LWM2M.
HomeKit	**These guidelines** make it easy for people to use home automation apps on their iOS devices to control and configure the connected accessories in their homes, regardless of the accessory manufacturer. HomeKit supports a number of consumer IoT-facing capabilities for the home: • Set up homes, rooms, and zones • Add, find, and remove accessories, such as light bulbs or thermostats • Define behaviors that apply to a set of multiple accessories • Manage users • Use Siri to control their homes Homekit defines strong encryption requirements for both WiFi and Bluetooth-LE communications, including 3072-bit keys and Elliptic Curve 25519 for signatures and key exchange.
IoTivity	The Open Interconnect Consortium sponsors the IoTivity open source project. IoTivity is an open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the IoT. There are options within the process for onboarding devices: 1) Just Works, 2) Random PIN, 3) Certificate IoTivity devices include a Security Resource Manager (SRM) responsible for filtering resource requests based on policy, and providing security management functions that include maintaining credentials and managing access control lists. The IoTivity JAVA API document can be found **here**. Additional details on IoTivity security can be found **here.**
ThingWorx	ThingWorx provides starter kits for popular IoT platforms such as the Raspberry Pi. ThingWorx provides secure connectivity between devices, device management, monitoring and control and remote data collection from devices. Monitoring support for events such as successful and failed logins and logouts. Also supports various authentication types (http basic, custom, token, etc). Supports user groups (administrators, designers, developers, guests, security administrators, users) and permissions. ThingWorx also integrates directly with Amazon Web Services IoT and Microsoft Azure Cloud.
Xively	Xively supports multiple protocols for communication, including HTTP, WebSockets, MQTT and mandates the use of TLS over each of these channels to achieve end-to-end security. Xively allows developers to connect and manage devices, and share data across an IoT ecosystem. Xively provides REST APIs that support capabilities including device management. Device, application and service authentication is done with API Keys in Xively. REST communications can and should be protected using TLS. Xively offers a number of TLS crypto suite options depending on the capabilities of an IoT device. Within Xively, devices with cryptographically secure random number generator (CRNG) can establish a standard secure TLS connection with the ECDHE-RSA handshake protocol. In this case, OCSP stapling ensures the validity of the server side certificates. In the case where the device does not support nonce-based OCSP, an accurate clock is necessary to ensure certificate validity checks.

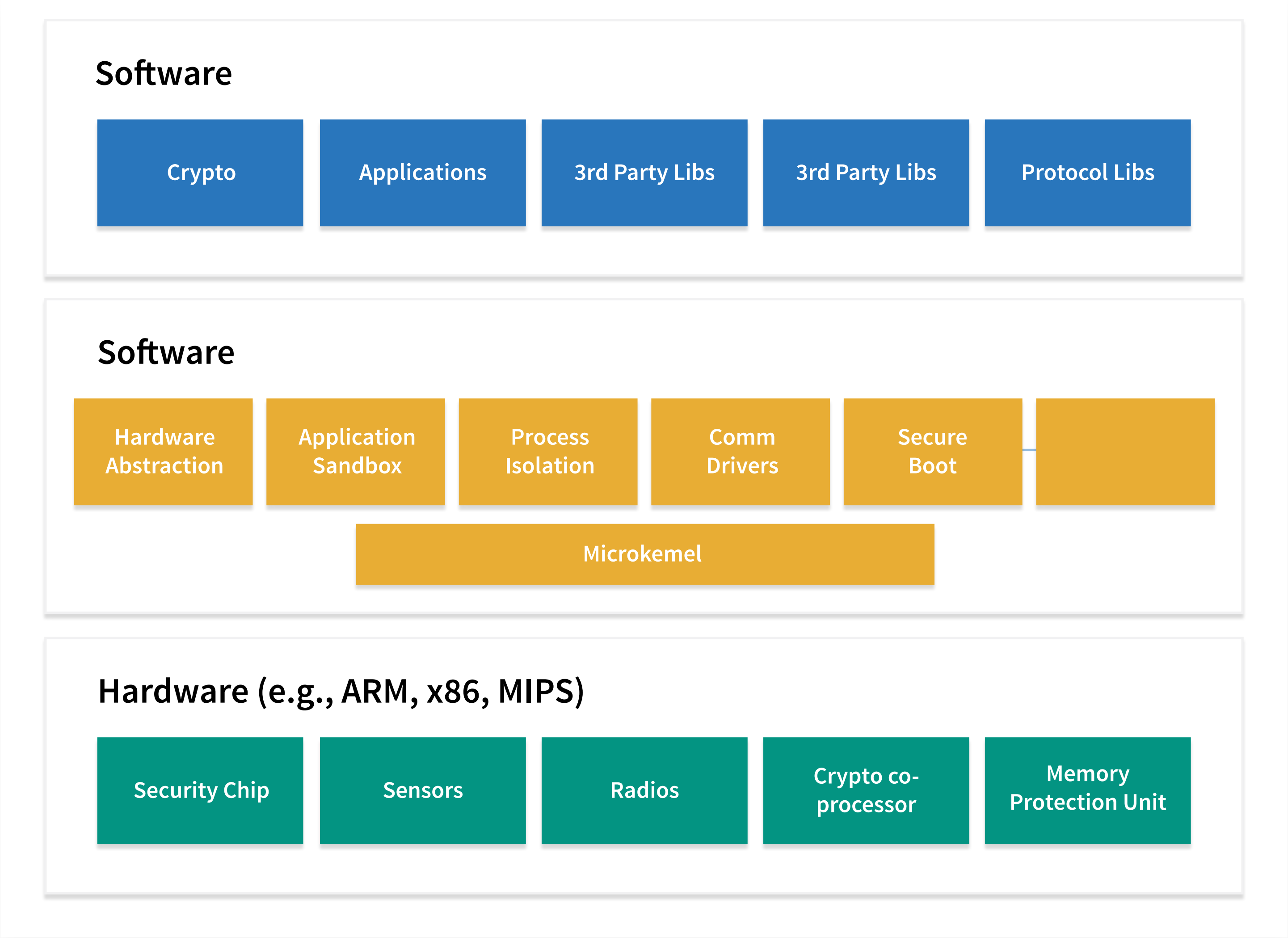
Table 6

	Devices with pseudorandom number generator (PRNG) and writeable memory can establish the same standard secure TLS connection. This requires having an initial cryptographically random seed embedded on the device. For these devices, Xively provides a software-based CRNG, where the PRNG state is regularly saved. Devices without any of these capabilities can connect to Xively using the TLS-PSK standard, which only requires a unique cryptographically random credential to be embedded on the device. This mode of operation typically lacks the security advantages that the certificate-based modes offer.
Oracle Java Embedded	Oracle Java ME Embedded supports operations on resource-constrained devices. Java ME Embedded Profile provides a security model based on protection domains. Enables application signing and supports a configurable security policy for devices. Learn more about the security model **here**.

Evaluate Platform Security Features

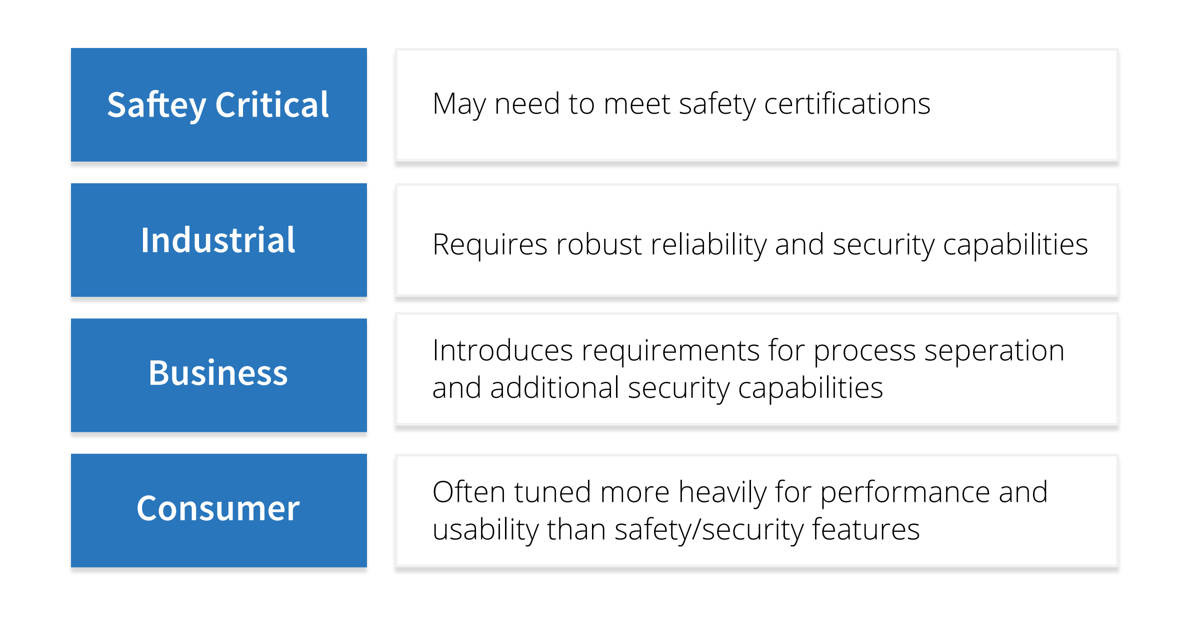
As you begin to design the technology architecture of your device, consider the various security features offered by each hardware and software component. Figure 2 provides a view into the technology layers of a typical IoT product. There are opportunities to evaluate the security features at each of these layers in order to create a defense-in-depth based architecture to secure your product.

Figure 2



In addition to micro-hardware security protections, where possible, the use of secured operating systems are warranted. Many IoT device profiles are shrinking to small but powerful SoC units capable of running a Real Time Operating System (RTOS) which may include a variety of secured-boot Operating Systems featuring strict access controls, trusted execution environments, high security microkernels, kernel separation and other security features. Also note that different categories of IoT devices will require different RTOS solutions as outlined in Figure 3.

Figure 3



Saftey Critical	May need to meet safety certifications
Industrial	Requires robust reliability and security capabilities
Business	Introduces requirements for process seperation and additional security capabilities
Consumer	Often tuned more heavily for performance and usability than safety/security features

At the top end of the spectrum (Safety-critical IoT devices), RTOS selection should be based heavily on whether there is a need to meet industry-specific certifications. Examples of these certifications include [21]:

Table 7

Certification	Details	
DO-178B	Software Considerations in Airborne Systems and Equipment Certification for avionics systems	Link
IEC 61508	Functional Safety for industrial control systems	Link
ISO 62304	Medical Device Software - Software Lifecycle Processes, for medical devices	Link
SIL3/SIL4	Safety Integrity Level for transportation and nuclear systems	Link

A well written **2010 IEEE paper** by Wei Dong and Chun Chen provides a solid understanding of considerations related to operating systems designed to support wireless sensors. Although not all IoT devices will face the same challenges that WSN-based components will face, a good number of IoT devices will have a WSN-style profile (battery operated, limited processing capacity and limited or no storage space. The paper by Dong and Chen outline a set of typical functions provided by an RTOS:

- Task scheduling
- Dynamic Linking and Loading
- Resource abstraction
- Sensor interfaces
- Hardware abstraction

There are highly robust RTOS available, as an example from LynxOS and Green Hills Software that should be considered when dealing with safety-critical IoT systems. These are commonly referred to as Cyber-physical systems.

There are many other operating systems suitable for IoT devices. Table 8 provides a view into many of these choices. Note that many of these RTOS have restrictions as to which types of platforms they can be installed upon.

Table 8

O/S	Description
TinyOS	TinyOS is an operating system designed for low-power embedded systems. It is not an OS in the traditional sense, whilst it is a programming framework for embedded systems and set of components able to build an application-specific OS into each application. TinyOS is written in the NesC language, a dialect of C and it supports an event-driven concurrency model based on split-phase interfaces, asynchronous events, and deferred computation called tasks (**reference link**).
Contiki	Supports a full IP stack with UDP, TCP and HTTP features as well as 6loWPAN and CoAP. Designed to operate in extremely low power systems. Includes link layer encryption for 802.15.4 communications.
Mantis	An embedded multithreaded operating system for wireless sensor platforms. Implemented in a lightweight RAM footprint that fits in less than 500 bytes of memory, including kernel, scheduler, and network stack. Supports remote management of sensors via remote update and remote login. **Supports energy efficiency through sleep mode.** *(sha, carlson, et al. MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms, ACM Digital Library).*
Nano-RK	Geared towards surveillance and environmental monitoring. Supports energy efficiency. Supports pre-emptive multitasking. Supports CPU and network bandwidth reservations. Provides support for networking. Runs on 2KB RAM and 18KB ROM.
LiteOS	Includes a hierarchical file system and wirelessly accessible shell (similar to Unix). Also provides a remote debugging system. Uses 10KB and designed to run on many connected machines.
FreeRTOS	Used across a large quantity of microcontroller/microprocessor types. Includes the ability to add TCP networking features. May also support SSL/TLS security for communication links. Cryptographic libraries such as Wolf SSL have been ported for use on FreeRTOS.
SapphireOS	Supports mesh networking and device discovery. Comes with python tools loaded and a RESTful API server.
BrilloOS	Requires 32 to 64 MB RAM, originally designed/optimized for home-based (e.g., consumer) IoT devices.
uCLinux	Embedded micro linux. Includes a collection of user applications, libraries and tool chains. Does not support multi-tasking. Learn more about uCLinux **here.**

Table 8

ARM mbed OS	For IoT devices that will make use of ARM-based processors, mbed OS now offers uVisor. uVisor is the supervisory kernel of mbed OS that supports creation of isolated security domains on Cortex M3, M4 and M7 MCUs with a Memory Protection Unit (MPU) (**reference link**). Application developers can make use of uVisor to securely store encryption keys.
RIOT OS	Can run on 8, 16 and 32-bit platforms. Provides a TCP/IP networking stack and supports protocols used within the IoT such as 6LoWPAN, IPv6, UDP and CoaP. Supports C and C++. Supports multithreading. Requires 1.5 KB RAM and 5 KB ROM.
VXWors	From Wind River. Two version: VXWors and VXWors+. Optional add-on security profile includes features such as secure partitioning, secure boot, secure run-time loader and advanced user management. Also includes network security features and encrypted containers.
LynxOS	Supports a number of networking features including TCP/IP, IPv6 and cellular (2G, 3G, 4G) communications. Also supports 802.11 WiFi, ZigBee and BlueTooth. Features various security features including robust encryption support, access controls, auditing and account management.
Zephyr	Designed for resource-constrained systems. Open-source project with a heavy focus on secure development. Implements a nano-kernel as well as a micro-kernel. Supports Bluetooth, Bluetooth-LE and 802.15.4 (6LoWPAN.
Windows 10 IoT	Supports bitlocker encryption and secure boot. Also includes DeviceGuard and CredentialGuard features. Supports updates through the Windows Server Update Service (WSUS).
QNA	An operating system often used in vehicular infotainment systems. Offers various security features including the ability to launch application in a sandbox and fine-grained control of system privilege levels.
Ubuntu Core	Read-only root filesystem. Security sandbox for applications. Supports separate (independent) update of applications from the OS. Supports ability to categorize applications as trusted or untrusted (by OS). Supports Unified Extensible Firmware Interface (UEFI) secure boot. Learn more about Ubuntu Core security features **here.**

Another note to consider is that many IoT devices are designed to optimize battery consumption (e.g., the OS may put the device to sleep on a regular basis and only wake when an activity needs to be accomplished). Consider that there are those out there (e.g., attackers) that would seek to drain your products' battery by keeping the device awake.

4. Establish Privacy Protections

The U.S. Federal Trade Commission (FTC) Chairwoman, Edith Ramirez, **cautioned** in January 2016 that the proliferation of IoT devices raises concerns about the personal information that is being collected, how it is being used and whether it is adequately secured.

In a detailed **report on the Internet of Things**, the FTC recommended a series of concrete steps that businesses can start taking to enhance and protect consumers' privacy and security:

- build security into devices at the outset, rather than as an afterthought in the design process;
- train employees about the importance of security, and ensure that security is managed at an appropriate level in the organization;
- ensure that when outside service providers are hired, that those providers are capable of maintaining reasonable security, and provide reasonable oversight of the providers;
- when a security risk is identified, consider a "defense-in-depth" strategy whereby multiple layers of security may be used to defend against a particular risk;
- consider measures to keep unauthorized users from accessing a consumer's device, data, or personal information stored on the network;
- monitor connected devices throughout their expected life cycle, and where feasible, provide security patches to cover known risks; and
- consider data minimization, i.e. limiting the collection of consumer data, and retaining that information only for a set period of time, and not indefinitely.

Likewise, the European Union's Article 29 Working Party (WP29)issued an **opinion** in 2014 detailing how EU data protection rules apply to IoT. The WP29 is an advisory group composed of representatives from the data protection authorities of each EU member state. As such, while the opinion is not binding law, it is nevertheless persuasive and is a good indication as to how EU regulators will decide a particular issue. The WP29's opinion provides some **guidance** on privacy issues to be considered by IoT manufacturers, developers and data collectors. Recommendations include:

- A privacy impact assessment (PIA) should be performed before any new device or application is launched.
- IoT stakeholders typically need aggregated data only and therefore, raw data should be deleted from the device as soon as the data required for the data processing has been extracted.
- Follow privacy-by-design and privacy-by-default principles.
- Data subjects and users must be able to exercise their rights and thus be "in control" of their data at any time. Under EU data protection rules, users must be informed about the type of data that are collected and further processed, the types of data that will be received and how it will be processed, used and/or combined.
- Provide a privacy notice which is understandable by users and obtain the user's consent or offer the right to refuse. Consent to the use of a connected device and to the resulting data processing must be informed and freely given.
- Design devices to inform both users and people who would br interacting with them (e.g., people being recorded by a camera in a wearable technology) of the data processing which would be carried out by the entity providing the device.
- Inform users of data that has been collected and enable them to access, review and edit that data before it is transferred.
- Give users granular choices on the type of processing as well as time and frequency of data gathering. App developers and device manufacturers should provide an adequate level of information to end users, offer simple opt-outs and/or granular consent, when applicable. Furthermore, when consent has not been obtained, the data controller should anonymise the data before repurposing it or sharing them with other parties.

The WP29 opinion proposes that the methodology to be followed for such PIAs can be based on the **Privacy and Data Protection Impact Assessment Framework** which the WP29 has adopted on 12 January 2011 in relation to RFID Applications. If it is found that a device collects, processes or stores private personal information(PPI), more stringent controls will be required. These controls should be a mix of policy-based and technical measures. For example, the provisioning of a device may require certain administrative approvals. A PIA should be conducted to determine if it is necessary to have PPI data stored on IoT devices. Data stored on the device should be encrypted using sufficiently strong cryptographic algorithms Data transmitted from/to the device should be encrypted using sufficiently strong cryptographic algorithms Access to the device, both physical and logical, should be restricted to authorized personnel.

Understanding the risks exposed by an IoT device is a complicated matter and device manufacturers have the responsibility to understand the types of data that will be processed or stored on the devices as well as the data that is transmitted to the services and mobile applications that connect to the devices. This understanding provides the foundation for designing in security controls during development that allow for organizations to protect their information assets. Consider the following additional design guidance during a device development.

Design IoT devices, services and systems to collect only the minimum amount of data necessary

This may seem straightforward, however there are opportunities to get things wrong related to data collection. IoT device manufacturers should review data models with an eye towards privacy. To this end they should:

- Reduce at minimum the stored data
- Avoid data leakages

Indeed, sensors may inadvertently leak information about someone that interacts with the device, or where capture of device data could allow someone to infer characteristics about someone.

A good example is the case of a smart meter. If someone gains access to smart meter data (energy usage history for example), can that person then infer the behavioral pattern of the homeowners?

IoT device don’t operate in a vacuum, they are part of larger systems. Device manufacturers should also be cognizant of identifying data processed by the device that, when paired with data collected by other devices, could leak sensitive information. Following these guidelines is of particular importance especially for those applications that manage sensitive data, including heath, ….(some more reference scenarios here) applications.

Also, Design IoT devices, services and systems to support anonymity when possible. Information that can tie back to an individual or entity should be closely evaluated for potential anonymization approaches. As an example, manufacturers of connected vehicles and components that are used within connected vehicles must take great care to ensure that the identify of the driver cannot be determined based on any of the data collected and then transmitted.

Note also that given the state of the IoT industry, new use cases for IoT devices will likely sprout up all the time. Your product may be used in entirely new ways than initially conceived after hitting the market.

Analyze device use cases to support compliance mandates as necessary

When dealing with IoT and Cyber-Physical Systems (CPS), regulatory and compliance challenges are likely to grow in their importance. As the National Institute of Standards and Technologies (NIST) appointed (**reference link**):

"smart systems are co-engineered interacting networks of physical and computational components. These systems will provide the foundation of our critical infrastructure, form the basis of emerging and future smart services, and improve our quality of life in many areas."

These advances bring with them several regulatory and Standard compliance challenges. This is the key-factor - for example - for devices related to health (e.g., wearable devices). Indeed, such devices may not only perform calories counting or track fitness levels, but they may potentially infer more sensitive information regarding wearer health conditions. In this case devices may be required to be compliant with the Health Insurance Portability and Accountability Act (HIPAA). Other compliance challenges may rise for those devices designed to be used by children under 30 (Children's Online Privacy Protection Act - COPPA).

More generally, privacy and data security gained much attention from the compliance side, especially in 2015. Just to mention some:

- **EU Data Protection Directive**, for devices sold and/or used in the European Union.
- **EU General Data Protection Regulation** (upcoming), for devices….(to be completed).
- **EU-US Privacy Shield**, if it envisaged the transfer of personal information from EU to US.

Does the device likely need to meet any specific compliance requirements? For example, if the device relates to health (e.g., a wearable), does it comply with the Health Insurance Portability and Accountability Act (HIPAA) in the United States? If the device is meant to be used by children under 13, does it comply with the Children's Online Privacy Protection Act (COPPA) in the United States? If the device is sold or used in the European Union, does it comply with the EU data protection rules (e.g. the EU Data Protection Directive, the upcoming EU General Data Protection Regulation, the EU-US Privacy Shield if a transfer of personal information from the EU to the US is envisaged, etc.).

Design opt-in requirements for IoT device, service and system features

Many IoT devices are headless, in that there is no HMI to provide opt-in agreements to users. Even IoT devices that have minimal screens would provide an inconvenient amount of space for displaying these agreements. IoT device developers nonetheless should think of methods for allowing users to opt-in to storage and sharing of private information. This opt-in should be as granular as possible.

Providing these opt-in options on devices that interface with the IoT device, for example smart phones, would be the logical choice for the consumer market. For IoT devices geared towards business use, developers would be well served to provide data sheets that describe all of the data collected. These data sheets can be used by enterprise organizations that deploy the devices to inform their stakeholders and allow for opt-in through out-of-band methods.

Implement Technical Privacy Protections

Privacy-enhanced Discovery Features

Ensure that bluetooth devices are not designed to support publicly discoverable static MAC addresses. Instead, implement support for BLE privacy features introduced in 4.2 of the specification, that include the ability to generate resolvable MAC addresses cryptographically during the pairing process.

Rotating Certificates

In some instances, it is not acceptable to be able to tie any information back to the owner of an IoT device. In the case of Connected Vehicles, there is a scheme that supports the constant rotation of a pool of certificates /key pairs used for digitally signing transactions. Consider an approach such as this when privacy requirements are absolute.

5. Design in Hardware-based Security Controls

The IoT forces product developers to evaluate and implement hardware protection mechanisms within their products. Software vulnerabilities and misconfigurations are not the only attack vector with which to be concerned. Indeed, there are things to consider related to securing the hardware of your IoT products that include:

1. There are tools available that can extract data (including firmware) directly from internal hardware components
2. Attackers can identify internal hardware debug interfaces that may be used to gain access even if they are not labeled
3. If firmware is extracted attackers may be able to upload modified firmware to change the intended behavior of the product

IoT devices are often a mixture of hardware and software, however the combination of components can be as simple as a microcontroller (MCU) and sensor paired together on a Printed Circuit Board (PCB). Other implementations may be far more complex and include a number of hardware ports (e.g., JTAG) that may be left exposed and unprotected for the intended environment.

The MicroController (MCU)

One of the first choices to make is to determine which Microcontroller (MCU) will be used. The leading architectures for IoT today are ARM, MIPS and x86. For detailed descriptions of hardware security mechanisms please read the report released by the PRPL foundation - **Security Guidance for Critical Areas of Embedded Computing**.

Selection of an MCU for an IoT development is a typical starting point for technology selection processes. The selection of an MCU is heavily based on the functional requirements of the IoT device, as MCUs that offer support for low-power applications, performance applications, and even wireless applications are all available. These System on Chip (SoC) solutions provide many of the core capabilities that some IoT devices require. As an example, a SoC solution may provide an MCU with a Near Field Communication (NFC) transponder that is tightly integrated onto a single platform.

Although some IoT devices are more complex, many sensors are significantly constrained in terms of both energy and computational power, thus requiring only minimal additional technology components on top of the chosen SoC solution. Either way, the selection of the SoC foundation for your IoT device development is a crucial security consideration. The following should be considered when choosing a SoC:

1. Does the SoC offer a cryptographic bootloader [20] that can be leveraged to support secure firmware updates?
2. Does the SoC offer a cryptographic hardware accelerator to support efficient cryptographic processing, and what algorithms are supported by the accelerator?
3. Does the SoC offer secure memory protection
4. Does the SoC offer built-in tamper protections (e.g., JTAG security fuses)
5. Does the SoC offer protection against reverse engineering?
6. Does the SoC offer secure mechanisms for cryptographic key storage in nonvolatile memory?

In a traditional Information Technology (IT) sense, hardware security protections come in the form of things like Hardware Security Modules (HSMs) and Trusted Platform Modules (TPMs). Although these devices may still have a role to play in an overarching IoT ecosystem, at the device level they have limited applicability (based on resource constraints and cost considerations). However, some robust IoT platforms may support capabilities that include TPMs. This section will examine some of the options that designers and developers of IoT devices can consider as they plan for using security components to protect a device. We will also look at some of the options available for protecting an IoT product from malicious physical access.

Some technologies also support the separation of hardware into trusted and non-trusted zones. Technology approaches such as ARM's Trust Zone offer this capability to provide features such as sandboxing, firmware protection, secure root of trust and code isolation. Figure x provides a view of the ARM TrustZone separation.

Trusted Platform Modules

Although this is not always possible, consider incorporating a Trusted Platform Module (TPM) into your IoT device to safeguard the device's cryptographic boundary. Although this can increase costs and is often not achievable on constrained IoT devices, strongly consider this option when creating a device that processes or stores sensitive information. A TPM can be used to extend the zone of trust to other portions of the design to insure they are not compromised. The TPM does this by authenticating and authorizing transmissions to and from the system using known standards for encryption, decryption and authentication (**reference link**).

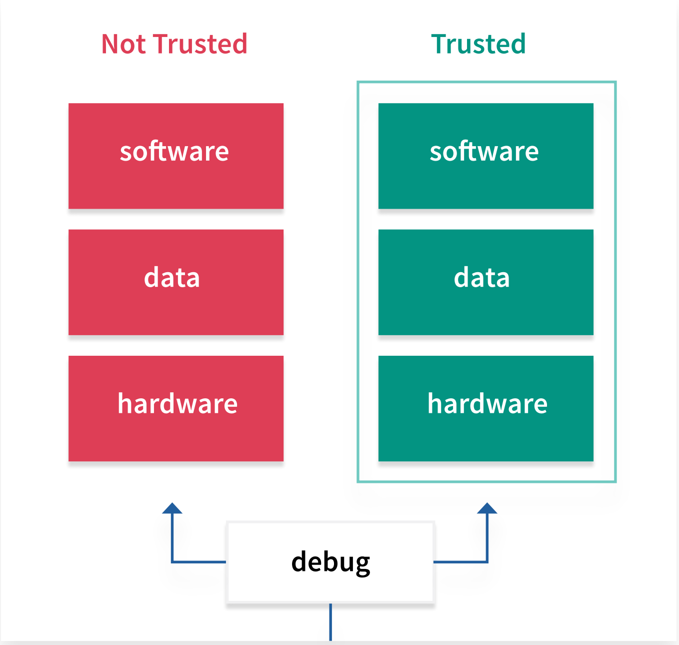


Figure 4

Use of Memory Protection Units (MPUs)

Some MCU developers include optional memory protection units (MPUs) that can be integrated with the MCU. MPUs provide access rules to memory locations, allowing IoT products that incorporate an MPU to control what memory can be read, written and executed.

Incorporate Physically Unclonable Functions

Physically Unclonable Functions (PUFs) are information security devices that capture the value of deriving an identity from a physical attribute of an object (reference: R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, Physical one-way functions, Science 297, pp. 2026–2030, 2002). The identify is easy to evaluate but hard to predict. Researchers such as Robert Young are doing interesting and useful work by adapting the concept of PUFs to the nanoscale. Dr. Young's research in Quantum-Confinement PUFs (QC-PUFs) derives an identity from information at the molecular level of a device. They provide a toolkit that can be used on a number of platforms.

Devices such as the SmartFusion Field Programmable Gate Array (FPGA) from Microsemi can provide these features (**reference link**).

Use of specialized security chips/coprocessors

Many MCU vendors now market specialized security chips that can be used to securely store credentials and other cryptographic data on IoT devices. Wikipedia provides a good list of features that should be offered by security coprocessors, including:

- Tamper-detection and and tamper-evidence
- Conductive shield layers in the chip that prevent reading of internal signals.
- Controlled execution to prevent timing delays from revealing any secret information.
- Automatic zeroization of secrets in the event of tampering.
- Chain of trust boot-loader which authenticates the operating system before loading it.
- Chain of trust operating system which authenticates application software before loading it.
- Hardware-based capability registers, implementing a one-way privilege separation model.

Table 9 provides a view into some of these MCUs on the market today. Note that you must evaluate each solution with your planned IoT product hardware.

Table 9

Vendor	Product	Details
NXP	A710X	This family features a dedicated MX51 security CPU with an optional X.509 certificate-based authentication module installed and on-chip cryptographic library. Includes an AES co-processor and active shield technology. Read more about this MCU family **here**.
Atmel	Crypto Authentication	Provides hardware-based key storage. Supports secure download/ boot. Supports anti-cloning and message security. Learn more **here**.
Atmel	CryptoMemory	High security EEPROM. Supports separation of memory into 16 different sections. Learn more **here**.
Atmel	ATECC508A	Cryptographic coprocessor.
microchip/ ATMEL	CEC1302	Provides key generation and authentication functions. Supports cryptographic acceleration.

Table 9

ST Micro-electronics	(ST32G512A / ST33G1M2A)	Geared towards connected vehicle market. Provides an embedded secure element (for a root of trust), supports authentication.
Freescale	K80	Includes an MMU; includes hardware implementation of cryptographic algorithms (AES; SHA-256, etc); supports encrypted firmware updates
Infineon	OPTIGA TRUST P SLJ 52ACA	Supports multiple functions: authentication, secure boot, secure update, key generation/storage, protected memory

Spend time considering threats to the underlying IoT device hardware and firmware, and incorporate requirements into the design as well as procedures for procurement across your supply chain- to minimize those threats. As you design your IoT device, it is helpful to identify security boundaries that require extra attention and safeguards. For example, identifying the cryptographic boundary of the device will provide a view into the areas of the hardware/software that must be provided with the most significant degree of protections. Inside of the cryptographic boundary, you would find functions such as cryptographic processing and key management.

Use of cryptographic modules

It is helpful to identify security boundaries that require extra attention and safeguards. For example, identifying the physical cryptographic protections of the device will provide a view into the areas of the hardware/software that must be provided with the most significant degree of protections. Its utilization of cryptography and overall cryptographic hygiene are also important. Inside of the cryptographic boundary, one should find functions such as cryptographic processing and key management that are afforded additional security protections to guard against exposure of cryptographic key material and other sensitive security parameters.

The National Institute of Standards and Technology (NIST) provides valuable documentation and tools for secure cryptographic modules. The Federal Information Processing Standard (FIPS) 140-2 should be followed whenever implementing cryptographic protections within an IoT device. IoT developers can procure FIPS 140-2 validated modules or create their own to be certified modules. The FIPS 140-2 Security Requirements for Cryptographic Modules document provides a valuable summary of the requirements that span the most lenient (Level 1) to the most stringent (Level 4) design requirements for cryptographic modules.

Table 10

	Security Level 1	Security Level 2	Security Level 3	Security Level 4
Cryptographic Module Specification	Specification of cryptographic module, cryptographic boundary, approved algorithms, and approved modes of operation. Description of cryptographic module, including all hardware, software, and firmware components. Statement of module security policy.			
Cryptographic Module Ports and Interfaces	Required and optional interafes and of all input and output data paths.		Data ports for unprotected critical security parameters logically or physically seperated from other data ports.	
Roles Services and Authentication	Logical seperation of required and optional roles and services	Role-based or identity-based operator authentication	Identity-based operator authentication	
Finite State Model	Specification of finite state model. Required states and optional states. State transition diagram and specification of state transitions.			

Table 10

Physical Security	Production grade equipment	Locks or tamper evidence	Tamper detection and response for covers and doors	Tamper detection and response envelope. EFP or EFT.
Cryptographic Key Management	Key management mechanisms: random number and key generation, key establishment, key distribution, key entry/output, key storage, and key zeroization.			
	Secret and private keys established using manual methods may be entered or output in plaintext form		Secret and private keys established using manual methods shall be entered or output encrypted or with split knowledge procedures.	
EMI/EMC	Secret and private keys established using manual methods may be entered or output in plaintext form		Secret and private keys established using manual methods shall be entered or output encrypted or with split knowledge procedures.	
Self-tests	Power-up tests: cryptographic algorithm tests, software/firmware integrity tests, critical functions tests. Confitional tests.			
Design Assurance	Configuration management (CM) Secure installation and generation. Design policy correspondence. Guidance documents.	CM system. Secure distribution. Functional specification.	High-level language implementation.	Formal model. Detailed explanations (informed proofs). Preconditions and postconditions.
Mitigation of other attacks	Specification of mitigation of attacks for which no testable requirements are currently available.			

Device Physical Protections

Research related to an IoT device known as the Ring Video doorbell can also provide an interesting view of the need for IoT device manufacturers to consider defense-in-depth measures. Because the product was meant to be installed outside a home, it was possible for researchers to dismount the product and then view plaintext configuration data, including the Pre-Shared Key (PSK) of the home network that it is configured to operate on. The ability to dismount the device was facilitated by using regular mounting screws that could easily be removed, paired with applying no confidentiality to configuration data. Consider options for applying physical security protections, to include extremes such as tamper detection and even zeroization in extreme cases.

In a recent IoT security incident attackers used reverse engineering technique to analyze hardware and firmware to find vulnerability and/or way to modify to disable security features. Some critical use devices may need to be considered for protection from reverse engineering to prevent acquisition, analyze, modification of firmware. For example, encryption of firmware update package may prevent acquisition. Tamper proof hardware is the best way, although the costs of implementing in large scale IoT products may be prohibitive.

Tamper Protections

Anti-Tamper Protections provide safeguards against the compromise of sensitive information stored on the IoT device. Anti-tamper can be implemented in ways that alert (tamper-evident) to an attempted (or successful) physical compromise of the device, or actively defend against (tamper-resistant) that compromise.

Warren Miller from IHS Electronics360 recently provided good advice for implementing tamper protections:

"One easy trick to detect attacks on your PCB is to add extra signals from one MCU output port to an input port. These signals can snake around the board and can be placed over and under critical signals a hacker might attempt to gain access to. Any attempt to cut the board to access these signals will break a tamper detection signal and will alert the hardware that an attack is underway. The hardware can respond with a variety of countermeasures from resetting the system to erasing critical information such as code or passwords" (**reference link**).

Guard the Supply Chain

There are multiple considerations when discussing the safeguarding of the supply chain for IoT products. In some critical infrastructure systems, it is not unwise to keep an approved product list (APL) that outlines what components and libraries are allowed to be integrated into the product. In these instances, each components or library is reviewed to analyze whether its use would add additional risk to the security of the product.

Additionally, it is recommended that all open source libraries used in IoT products be passed through the same security checks as custom developed code. Developers should also be alert to any patch updates to those libraries and make updates as soon as feasible to decrease the likelihood that a vulnerability in open source code will lead to a compromise of the IoT product.

Self-Tests

Implement self-tests upon startup that verify the security features of the device. Upon identification of an error, log the error and discontinue processing if feasible.

Secure Physical Interfaces

One of the goals associated with safeguarding physical interfaces is to secure against offloading product firmware for analysis and modification. In high assurance type implementations (e.g., FIPS 140-2 Level 4) there are requirements for active tamper protections that would render a device useless in the case of the compromise of the physical casing. Most IoT products do not have this security luxury and there are likely no safeguards such as active tamper built into the device's physical protection scheme.

This means that attackers (and researchers) can easily gain access to the internals of the device. Once this is accomplished, they can evaluate the internal components to determine if there are physical ports exposed that would provide them with unauthorized access to the device (e.g., firmware or command prompt). Two areas of concern within IoT products are exposed:

- Universal Asynchronous Receiver Transmitter (UART) pins,
- Joint Test Action Group (JTAG) interface

Using custom tools, attackers can connect to these interfaces. It is therefore important to include a step in the engineering process that disables the interfaces before shipment. Additionally, password restrictions can be placed on the interfaces.

IoT products may also have additional physical interfaces. When possible, lock down USB ports to only trusted connections. Consider how to manage the security of any 3rd party integrated I/O in the device.

Also consider whether the device connects to other devices (e.g., an upstream computer or smartphone). Can that interface be exploited by taking advantage of the trust relationship between the devices? Is the computer that the IoT product connects to actually trusted? Consider sending a notification to the operator that asks for consent to share data with a non-trusted device.

Finally, remove debug code from the operational product to limit the attack surface.

6. Protect Data

One of the primary challenges for IoT device developers is understanding the interactions between different types of IoT protocols, and the optimal approach for layering security across these protocols. There are many options for establishing communication capabilities for IoT devices, and often these communication protocols provide a layer of authentication and encryption that should be applied at the link layer. Examples of IoT communication protocols such as Zigbee, ZWave and Bluetooth-LE all have configuration options for applying authentication, data integrity and confidentiality protections. Each of these protocols support the ability to create wireless networks of IoT devices. ùWi-Fi is also an option for supporting the wireless link required for many IoT devices.

Riding above the IoT communication protocols are data-centric protocols. Many of these protocols require the services of lower layer security capabilities, such as those provided by the IoT communication protocols or security-specific protocols such as DTLS or SASL. IoT data centric protocols can be divided into two categories, that include REST-type protocols such as CoAP and publish/subscribe protocols such as DDS and MQTT. These often require an underlying IP layer, however some protocols such as MQTT-SN have been tailored to operate on RF links such as Zigbee.

One of the important points to note about the IoT is that the traditional internet (as known today) is more human-to-machine centric. However with the rapid adoption and evolution of the IoT, this will also need to include machine-to-machine transactions. This also would mean one more critical difference, there will not be a data steady flow (more likely in bursts) given the nature and usage of these devices. The data packet will also be very targeted and terse. The actual summarization and intelligence processing will be elsewhere upstream from the IoT, either in a gateway or in the cloud.

The IoT device will also have to be considered from an optimal cost perspective (sub $50 and sensors in the $1 or less range given the volumes that need to be considered). Given this context it may not make sense to factor a heavy protocol stack.

The data definition will also need to be looked at from an ease-of-manageability perspective on the It, and the ability to be support low data packet size and the terse nature of the content. Some options that can be considered are the following:

- **Apache Avro**
- **Apache Thrift**
- **Protocol buffer**

There are many IoT protocols to choose from and given the nature of the IoT, most of these protocols make use of UDP as the transport instead of TCP. Fault tolerance in the transport can be dealt with via normalization of the data stream. As an example, dealing with the loss of data for a few minutes from an environmental sensor that reports temperature data every few minutes can often be tolerated. This should be taken into account when selecting an appropriate protocol suite to employ for an IoT device, as the level of fault tolerance required depends heavily on the type of data being processed.

Security Considerations for Selecting IoT Communication Protocols

There are many options to choose from and the choice of communication protocol will obviously depend on the use cases for the IoT product. In today's environment, manufacturers often rely upon short range communications that are tailored heavily towards devices with limited battery, which often must go into sleep modes and then wake up to receive or transmit data.

Things are also changing quickly in regards to IoT communications protocols. Although cellular communications are not the most widely deployed across all IoT device types, that may change with the introduction of 5G cellular, which is being optimized to support higher bandwidths as well as the ability to connection millions of devices. This change could have significant effects on architectures for IoT systems in the future as a move away from a gateway-heavy approach may be achievable. With 5G and earlier forms of cellular communication, a SIM card is typically employed to securely store credentials /identities of the device. These sizes of these cards are being shrunk significantly enough to support use in many types of IoT devices.

There are a variety of attacks that IoT product developers must guard against when selecting communication protocols. Although the selected communication protocol(s) may not guard against all of these attacks, the concept of defense-in-depth should be applied to a product / system in order to implement additional safeguards as needed.

- Wired and wireless scanning and mapping attacks
- Protocol attacks
- Evesdropping attacks (loss of confidentiality)
- Cryptographic algorithm and key management attacks
- Spoofing and masquerading (authentication attacks)
- Denial of Service and jamming

IoT product developers should take time to gain an understanding of the various protocols available to them, however one protocol serves as the foundation (PHY and Data Link Layer) for various other protocols. 802.15.4 is the underlying standard for other IoT protocols such as ZigBee.

One of the key security considerations that should be examined for any protocol is the join or pairing process. For example, as a device is joining an existing network, what controls are put in place to ensure that only a legitimate device is able to join the network? Some protocols have options available such as "Just Works" that make adding devices to a network highly convenient, but also introduce security concerns related to the introduction of rogue devices onto home and business networks.

We can look at Bluetooth-LE for an example of the pairing process.

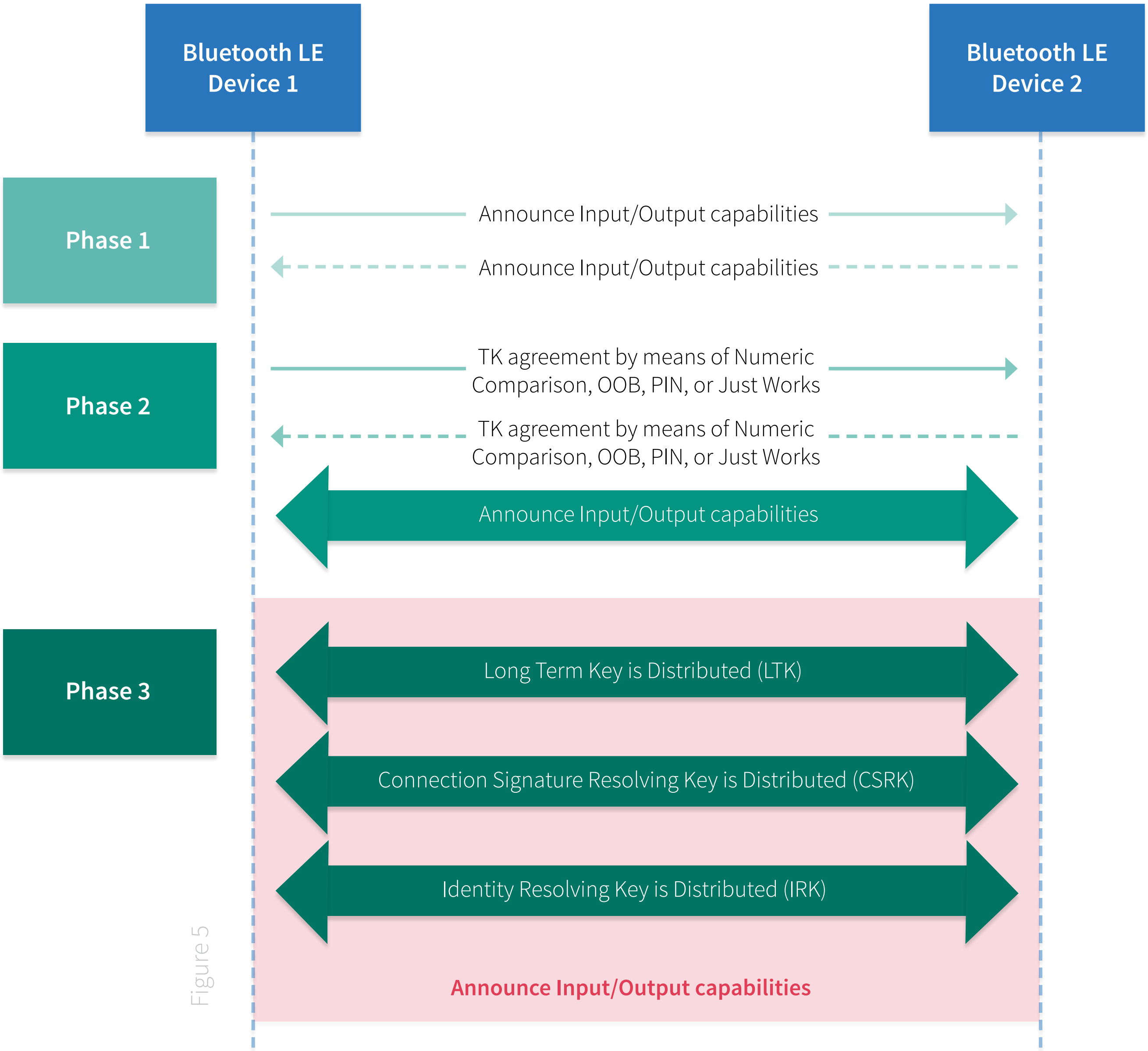

Bluetooth LE Device 1
Bluetooth LE Device 2
Phase 1
Announce Input/Output capabilities
Announce Input/Output capabilities
Phase 2
TK agreement by means of Numeric Comparison, OOB, PIN, or Just Works
TK agreement by means of Numeric Comparison, OOB, PIN, or Just Works
Announce Input/Output capabilities
Phase 3
Long Term Key is Distributed (LTK)
Connection Signature Resolving Key is Distributed (CSRK)
Identity Resolving Key is Distributed (IRK)
Announce Input/Output capabilities

Figure 5

What we can see within the Bluetooth-LE pairing process is that there are multiple keys that must be generated across 3 phases of the process. Phase 1 is negotiation of security capabilities to determine which modes to use for authentication and confidentiality. Phase 2 is the phase where the components are authenticated and this is where various mechanisms can be used, which include provisioning of PINs, out of band methods, or numeric comparison with ECDH key agreement. The Just Works option is easiest but should be avoided whenever possible.

Another security consideration is the maintenance of any trust relationship between devices or devices and applications, providing by a communications protocol. Does the communication protocol require an authenticated re-establishment of the trusted session?

Another consideration is the potential for a required gateway to be compromised. When communication protocols require gateways and the gateway is compromised, attackers can manipulate data prior to the data getting to the final destination (e.g., the cloud). In the case of consumer IoT, gateways may be smartphone apps or even WiFi routers.

Some protocols rely upon beacons for device discovery. They are often broadcast to all nodes within a certain range These messages are open to potential spoofing if there is no authentication applied to the message.

Table 11

IoT Communication Protocol	Security Discussion
LTE	An Authentication and Key Agreement (AKA) procedure serves as the foundation for authentication between a **User Equipment (UE- aka an IoT device)** and the core network. UE consists of the Mobile Equipment and the USIM, which securely stores authentication information. Within the core network, a Home Subscriber Server (HSS) and Authentication Center (AuC) manage information for user authentication. The carrier loads pre-shared symmetric key in the AuC (network-side) and the USIM (user-side). The mutual authentication of user and network results in a derived key (Access Security Management Entity – ASME) which is then used to derive additional encryption and integrity keys for the NAS, RRC Signaling and User Plane. The pre-shared symmetric key is loaded into USIM as manufacture time and into the network at subscribe time.
GPRS	All data and signals are encrypted using the GPRS Encryption Algorithm (GEA) at the Logical Link Control (LLC) Layer. Uses a SIM card to store identities/keys.
GSM	Cellular technology. Time Division Multiple Access (TDMA) based. IoT devices / handsets are provisioned with SIM cards to store the identity/secret keys. Potential issues that may arise include insertion of malicious base stations, weak keys (small number of bits) and cleartext transmission of keys across network.
UMTS	Signalling and user data are encrypted. 128-bit cipher key used. Uses KASUMI algorithm with two modes: encryption and data integrity protection.
CDMA	Cellular technology. Code Division Multiple Access (CDMA). No SIM cards used.
Long Range Wide Area Network (LoRaWAN)	Data rates supported between 0.3 kbps to 50 kbps. Per the Lora Alliance, LoRaWAN relies upon use of three keys: • Unique Network Key • Unique Application Key for end-to-end security at application layer • Device-specific key Read the **whitepaper** by Robert Miller of MWR Labs for a good discussion of LoRaWAN security
802.11	WiFi has been around for a long time. Consider that devices will need to be provisioned with the keys that gain them access to a WiFi network.
802.15.4	Allows for higher layer protocols to define the authentication/encryption/integrity controls.
6LoWPA	A low power wireless personal area network designed to support the automatic joining of devices to the network when possible. Requires the introduction of a LoWPAN Bootstrapping Server to provisioning bootstrap information to 6LoPAN devices. This is how security credentials and other information can be provisioned to the joining devices, A “secured” 6LoWPAN network includes an Authentication Server to support authentication mechanisms such as Extensible Authentication Protocol (EAP). 6LoWPAN Bootstrapping Server includes ability to track a blacklist.
ZigBee	IEEE 802.15.4-2003 defines Physical (PHY) layer and Medium Access Control (MAC) Layer. ZigBee

Table 11

	provides Network Layer which supports Star, Tree and Mesh topologies. ZigBee Security Services provide Key Establishment, Key Transport, Frame Protection, Device Management. End-to-end security is provided when source and destination have access to the same key. MAC layer is protected by AES (CTR or CCM mode). Weakness exists in the initial join of a ZigBee device to a network (e.g., when device has not been pre-configured for the network). During the initial join, a single key may be sent unprotected. This causes a vulnerability resulting in key being able to be obtained by any listening device.
ZWave	Keys include frame encryption key and data origin authentication key. Keys are not sent in the clear. More information on ZWave security analysis can be found **here**.
Thread	Built on top of 802.15.4 for PHY/MAC. Supports connection of up to 250 devices within a network. Supports AES encryption. Use a Password Authenticated Key Exchange (PAKE) protocol. 802.15.4 MAC layer utilizes a network key. Provisioning of a new device requires the provision of this network key wrapped in a Key Encryption Key (KEK). Joining network requires the use of a "Commissioner." New nodes joining the network use DTLS to authenticate which results in the creation of a pairwise key that can be used to encrypt network parameters. More information about Threat security can be found **here**.
Sigfox	Ultra Narrow Band (UNB) in the 915 MHz Range (US) and 868 MHz range (Europe). Uses device private keys for message signing. Limits on the number of messages that can be sent per device per day (140). Introduces anti-replay protections.
Bluetooth / Bluetooth-LE	Version 4 of the specification added Elliptic Curve Diffie Hellman (ECDH) for key exchange during the pairing process (Numeric Comparison Association Method). Bluetooth-LE uses a 128-bit symmetric key for confidentiality protection. Protocol provides multiple levels of security from highly limited to much more robust. New privacy enhancements in latest specification support rotating of the MAC addresses for enhanced privacy (vs. static MAC addresses).
NFC	Limited security protections. Often used in combination with another protocol.
Wave 1609	Prevalent in Connected Vehicle communication. Relies heavily on IEEE 1609.2 certificates which support attribute tagging.

7. Secure Associated Applications and Services

It is not sufficient to simply focus on securing a single aspect of the IoT. IoT devices operate as part of a larger ecosystem. Each integration point represents a potential new pathway into the systems that can be used to gain unauthorized access to information or control systems. Consideration must be given to ensuring that apps and services that are paired with IoT devices have been developed using secure development best practices.

Smartphone apps are often used to configure IoT devices, or to interact with IoT devices (e.g., to view video feeds, etc). Smart phone apps also provide gateway functionality in some instances to funnel data from the IoT device to the cloud. Smart phone application developers will want to make use of security credentials to enable authenticated and integrity protected communications to IoT devices. In some instances, developers should consider implementing Certificate Pinning to prevent Man-in-the-Middle attacks occurring within untrusted networks.

IoT product developers must also consider limitations on the privileges afforded to mobile apps. Limit the scope of ability that a mobile app can exert over any particular IoT device. For example, an app that controls functionality within an internet-connected vehicle, should not be allowed to affect controls that are not explicitly supposed to be remotely administered. Privilege access capabilities need to be considered for both the configuring of the IoT device and the applications interacting with them.

For comprehensive guidance on securing mobile applications, please view the CSA **Mobile Application Security Testing** (MAST).

IoT devices often interface with cloud services that tie together devices and services even across industries. These connections support and improve data collection, analytics, and processing between IoT components that belong to different owners which can sometimes enable fully threaded autonomous operations. There are exciting new applications being developed regularly as more organizations understand the pieces of the puzzle and begin to brainstorm new ways of doing things. As with mobile applications, cloud service developers must use rigor in securing their systems. Oftentimes, cloud services provide interfaces for MQTT, REST, and other communications coming from gateways or directly from IoT devices.

Developers for cloud services or those using cloud services should leverage standards processes and frameworks for securing internal and third-party services such as the CSA **Cloud Controls Matrix** (CCM).

8. Protect Logical Interfaces/APIs

One of the most important considerations when developing IoT products is interface security. There are many cloud services that IoT products may interface with, as well as custom-developed smart-phone applications and even peer IoT products.

APIs expose service providers to denial of service attacks when not properly safeguarded. Use techniques such as rate-limiting to guard against compromised IoT products trying to flood the service with requests.

Gateways should check for proper format of messages as well as verify that only allowed data types are passed. This will guard against the potential to insert malicious code into the API communications that could result in the compromise of an IoT cloud service. It is also important to validate schema.

Error handling should also be considered. Be careful not to provide responses that are too detailed. These responses are good for troubleshooting, however they also provide attackers with significant data points when trying to determine their course of action.

Guard against replay attacks through techniques such as embedding timestamps and/or counters into messaging structures.

Do not rely on the use of API key for your program/device as a primary means of security. Implement more robust authentication and authorization controls whenever possible. Also, keep API keys secure by limiting their exposure, to include exposure within commits to systems such as GitLab.

Additionally, encrypt all API communications. Use protocols such as TLS and DTLS depending on the whether TLS or UDP can be employed by the IoT messaging protocol. Employ certificate pinning and guard against transmission of sensitive information within GET requests.

Review the **OWASP REST Security Cheat Sheet** for good advice to follow regarding use of the REST in IoT products.

Consider the following as well:

1. What application layer APIs are exposed?
2. Are secure interfaces sensitive to the language domain they are written in? (I.e. never store passwords as String in Java, because Java String is immutable and cannot be zeroized/wiped from memory)
3. Frequently, interoperability issues necessitate reduced security. Can reduced trust relationships be exploited?
4. Is there a software agent that resides externally? Is it used for managing, monitoring or troubleshooting, etc? Where does it reside? What is the trust relationship? What is the security profile of the agent?
5. Does the device initiate connections with other devices or services, do they initiate connections with the device, or both?

Application Programming Interfaces (APIs) allow for direct connection to the cloud or to a mobile device/gateway. As an example, the new Amazon Web Services (AWS) IoT Service comes with an API to allow direct connection to their IoT Gateway. The API supports REST and includes a number of security-relevant operations (**reference link**).

The AWS IoT API also allows developers to setup rules for the handling of IoT data. For example, data can be:written to a dynamoDB table, a Kinesis stream, or result in the invocation of a Lambda function. Data can also be republished to IoT devices as another MQTT topic. The CreateTopicRule operation allows developers to specify the exact handling of IoT data.

An interesting aspect of the AWS IoT API, is that it only supports 2048-bit RSA and as of this publication, does not yet support the more IoT-friendly (i.e., more efficient) elliptic curve cryptography. Adopting efficient cryptography is a key factor when dealing with IoT, since in many cases IoT involves using energy constrained devices.

Microsoft also provides a series of **REST-based IoT APIs** for integration with the Azure IoT hub. For example, the Azure IoT hub REST API allows for creation of a device identity and provisioning of symmetric keys for authentication. Microsoft Azure also provides a messaging API for sending REST messages from an IoT device to the Azure hub and vice versa. Microsoft notes in their API documentation that the developer is responsible for ensuring the security of the connection. This involves supporting authentication of devices to cloud services (and vice versa), encryption and integrity protection of the connection (via TLS) as well as configuration of policies for secure communication, access and data transfer.

Implement Certificate Pinning Support

Certificate pinning within IoT firmware and mobile applications provide protections against attacks where the IoT device is configured to interface with a malicious server or proxy (Man-in-the-middle). This could have occurred if the Certificate Authority was compromised for instance, and a malicious certificate was issued. With certificate pinning, the application developer embeds the services' certificate or public key in the application/firmware itself. This leads to some lifecycle maintenance issues, however certificate pinning provides additional protections for IoT device communication security.

SSL pinning can provide significant security enhancements in the case that a CA is compromised, however it is important to keep in mind that you must protect the embedded certificate from being overwritten without authorization. This requires secure data storage for each platform, as well as the ability to securely update the credential if ever needed. OWASP has done a great job providing examples of **various language implementations** and a cheat sheet on **certificate pinning**.

9. Provide a Secure Update Capability

Insufficient security of firmware updates may allow a malicious person to modify legitimate firmware and upload new malicious firmware into the product. That malicious firmware may disable security controls, implement new features or provide a mechanism for exfiltration of data. Do not let your IoT product get modified in this manner.

Determining the strategy for updating firmware and software of your IoT product is one of the most challenging areas of concern. Considerations include things like determining whether the design should be active/passive or active/active. Is it possible to patch and swap in real time to a patched component. Can software or firmware be isolated in sandboxes to validate security before implementation.

Firmware must be protected end-to-end and the entire life-cycle must be considered. For example, does the initial firmware load happen at a secure facility, using secure processes. Have you designed in contingencies in case an update session is interrupted - meaning can your IoT product roll-back to a previous version to avoid outages?

Understand what permissions need to be associated with the update process as well. In this regard, who is responsible for kicking off the update? Is it the product owner or the backend device infrastructure. At times it is better to automatically update an IoT product as often users will fail to do so in a timely manner, potentially leaving the product exposed to a vulnerability. However, a consideration should also be whether an attacker can make use of the update process to perform a denial of service (DoS) attack on the device.

Regarding authenticating the update transaction, it is good to understand that authentication of the firmware should be end-to-end. This requires that the device have secure storage for a root of trust that can be used to validate a signature applied to the firmware within the infrastructure. This of course leads to a new area of concern, making sure that the keys used to sign the firmware updates within the infrastructure are secured. Make use of solutions such as HSMs for this task. Also, make sure that the update server itself has been configured securely and do not forget that if anyone has access to the firmware at any point in the development life-cycle they have the ability to introduce intentional (or unintentional bugs).   Use your CI environment as described in this paper to scrub your software updates for vulnerabilities (SAST) and common mistakes.

Taking things a bit further, has the cryptographic suite that is used to digitally sign the firmware been thoroughly vetted. Look for certifications such as FIPS 140-2 for crypto library implementations used for these purposes. Leverage these tools for provide an encrypted transport as well and ideally encrypt the actual firmware image also. This assumes secure storage on the device for decryption keys.   Also don't forget to write-protect the product to guard against unauthorized firmware modifications.

Regarding the choice of an update mechanism, consider **The Update Framework**.

10. Implement Authentication, Authorization and Access Control Features

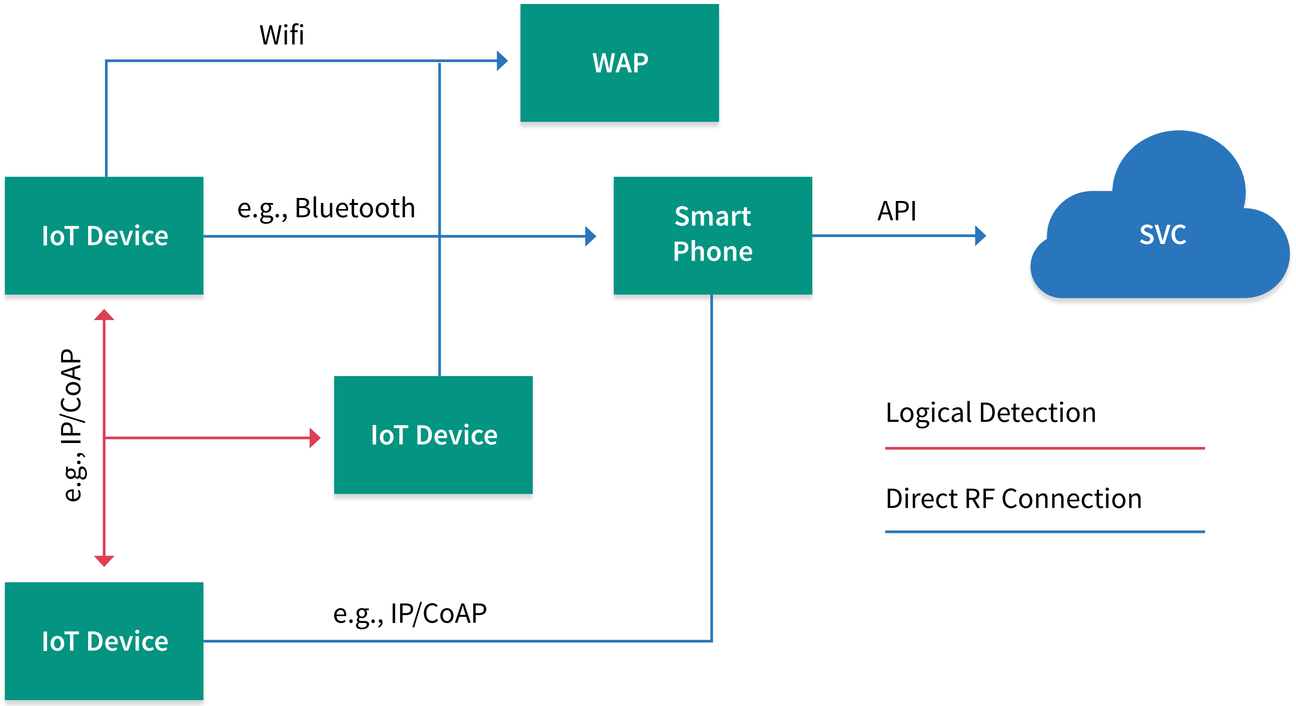
One of the key considerations to guard against in an IoT product development is credential theft. Whether this is the theft of passwords, access tokens or private keys, credentials must be safeguarded or the ability to restrict access is defeated. This means that developers must work on ways to implement secure storage, as well as keeping them from leaking in the case the device state is frozen and memory is probed. This document provides some recommendations for secure credential storage and use throughout.

One of the challenges associated with authentication and authorization in the IoT is the scale of devices involved for the end-user. Whether we are discussing dozens or hundreds of devices within a smart home, or hundreds of thousands to millions of devices for an enterprise, asking users to manually configure each device is a difficult proposition. The introduction of device to device communications complicates matters even more as many devices that will eventually work together will belong to different vendors and even ride on different frameworks.

As a developer of IoT products you will likely hear that today's identity management, authentication and authorization protocols and systems are not optimized for IoT solutions. If we take a step back to consider why this is true, we see that today's mechanisms are geared to support user identities. Even when devices such as smart phones are supported, they are typically attached to a user identity. With IoT devices, device identities may or may (likely not) not be associated with any particular individual user. Additionally, developers must provide customers with an ability to support administration (e.g., by enterprise technicians) of these devices. Sometimes the Original Equipment Manufacturer (OEM) may even require direct access to the devices to update firmware/software or configure devices.

When considering authentication, authorization and access control features, you must understand how your IoT products are used and managed. For consumer products, there is often a direct trust relationship between a user's smart phone and the device. In this instance, the smart phone application (IoT app) should authenticate to the IoT product and ideally vice versa. That's not the end of the discussion however. The power of the IoT is that devices can communicate with each other, preferably in an automated manner. This means that to enable a value-added IoT ecosystem (within a home environment or a business environment), IoT devices must be able to establish trust relationships with other IoT devices. This device-to-device communication must be established securely or you run the risk of a bad actor taking advantage of an entry point into the network.

Figure 6



Fortunately, many of the IoT protocols that support device-to-device communication come with the ability to configure secure connections. Table 12 provides a view into the configurations options available for some of these protocols.

Table 12

Protocol	M2m Authentication Options	Discussion
MQTT	username/password	MQTT allows for sending a username and password, although recommends that the password be no longer than 12 characters. Username and password are sent in the clear, and as such it is critical that TLS be employed when using MQTT.
CoAP	preSharedKey rawPublicKey certificate	CoAP supports multiple authentication options for device-to-device communication. Pair with Datagram TLS (D-TLS) for higher level confidentiality services.
XMPP	Multiple options available, depending on protocol	XMPP supports a variety of authentication patterns via the Simple Authentication and Security Layer (SASL - RFC4422) Mechanisms include one-way anonymous as well as mutual authentication with encrypted passwords, certificates and other means implemented through the SASL abstraction layer.
DDS	X.509 Certificates (PKI) Tokens	Provides endpoint authentication and key establishment to perform subsequent message data origin authentication (i.e., HMAC). Both digital certificates and various identity/ authorization token types are supported.
HTPP/REST	Basic Authentication (cleartext) (TLS methods) OAUTH2	HTTP/REST typically requires the support of the TLS protocol for authentication and confidentiality services. Although Basic Authentication (where credentials are passed in the clear) can be used under the cover of TLS, this is not a recommended practice. Instead attempt to stand up a token-based authentication approach such as OAUTH2

It is good to remember that authentication protections are best when they are end-to-end (E2E). The IoT often relies upon gateways that break up direct connections, so achieving E2E authentication protection is not always possible. Typically however, the higher you go in the stack (e.g., at the messaging level) the more likely to establish these E2E protections.

Designing multiple authentication options, layered within each other such as the TLS certificate-based authentication in addition to native protocol authentication, can offer end-to-end protections for your data.

Devices may also communicate directly with cloud services (without intermediaries). Many Cloud Service Providers (CSPs) are now offering gateways that support MQTT and REST communications. Device-to-cloud communications will often make use of API keys for the distinct services that are interacting with the device. This is in addition to the instantiation of TLS for protocols such as MQTT and REST communications, which can often implement two-way certificate-based authentication.

There are multiple methods for authentication and authorization that an IoT product developer may consider. In many cases, it makes sense to support multi-factor authentication. Putting together a solid authentication and authorization strategy for your product is based heavily on knowing the threat environment that their devices are intended to operate within. Is the planned authentication/authorization process adequate for the level of protection required within that environment? What specific threats against the device is the authentication/authorization scheme meant to mitigate. Be sure that you can answer these questions for all IoT products. Below we provide some thoughts on popular mechanisms for Authentication and Authorization considerations.

Using Certificates for Authentication

A certificate is bound to a cryptographic key pair (public/private). The public key is often embedded directly in the certificate and is made available to anyone that needs to validate the authenticity of the transaction. The private key is safeguarded, and in the case of authentication is used to digitally sign an object. A validation occurs when the associated public key is used to validate the signature applied to the object.

In order to gain a benefit from using this approach, developers must be sure that a secure Public Key Infrastructure (PKI) is available to issue certificates. Key pairs themselves should generally be generated on the IoT product however a PKI can issue the certificates based on receipt of a Certificate Signing Request (CSR). There are protocols available to aid this process, including the Simple Certificate Enrollment Protocol (SCEP) and the Enrollment over Secure Transport (EST) protocol.

Devices are authenticated based on their possession of the private key and trust in a 3rd party entity (the PKI).

Consider implementing two-way certificate authentication when possible. Standard X.509 certificates can provide a valuable layer of security for TLS communications as well as for protocol-specific secure messaging. Two-way certificate authentication allows the IoT product to pass its certificate for validation by the service or peer device/gateway it is communicating with. This is especially useful when dealing with protocols such as MQTT that support only username/password over cleartext.

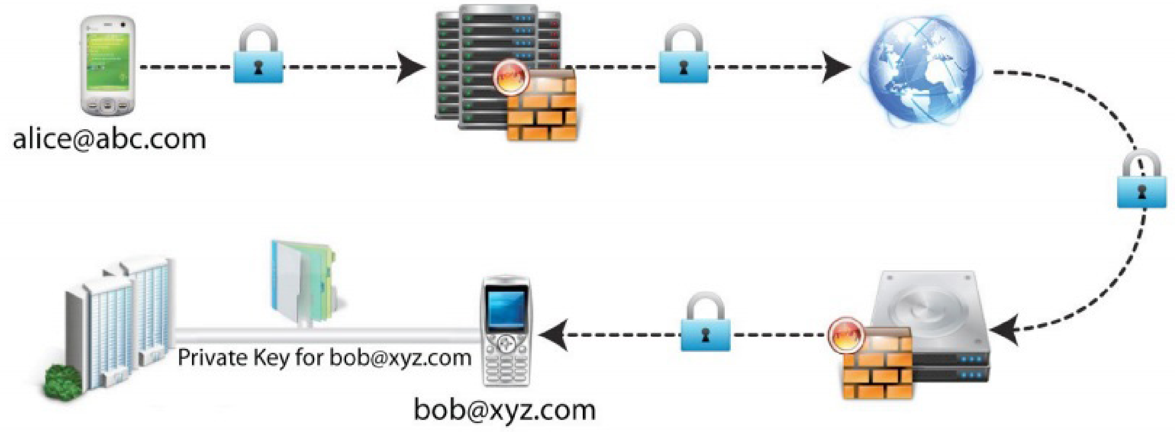
Consider Biometrics for Authentication

Biometrics can also be considered for use as a method for identification in authentication transactions. Biometrics such as fingerprints or even voice prints can and are being used to pass information about an entity (human) to be used in determining an authentication decision

Consider Certificate-Less Authenticated Encryption (CLAE)

A relatively new method for providing authentication within the IoT is known as Certificate-Less Authenticated Encryption (CLAE). CLAE allows the sender of a message to verify the public parameters of the server prior to encryption of the message. This allows for a verification that the public parameters used to encrypt the message have not been tampered with, in place of using certificates issued by a centralized Certificate Authority (CA). *Image below courtesy of Connect in Private.*

Figure 7



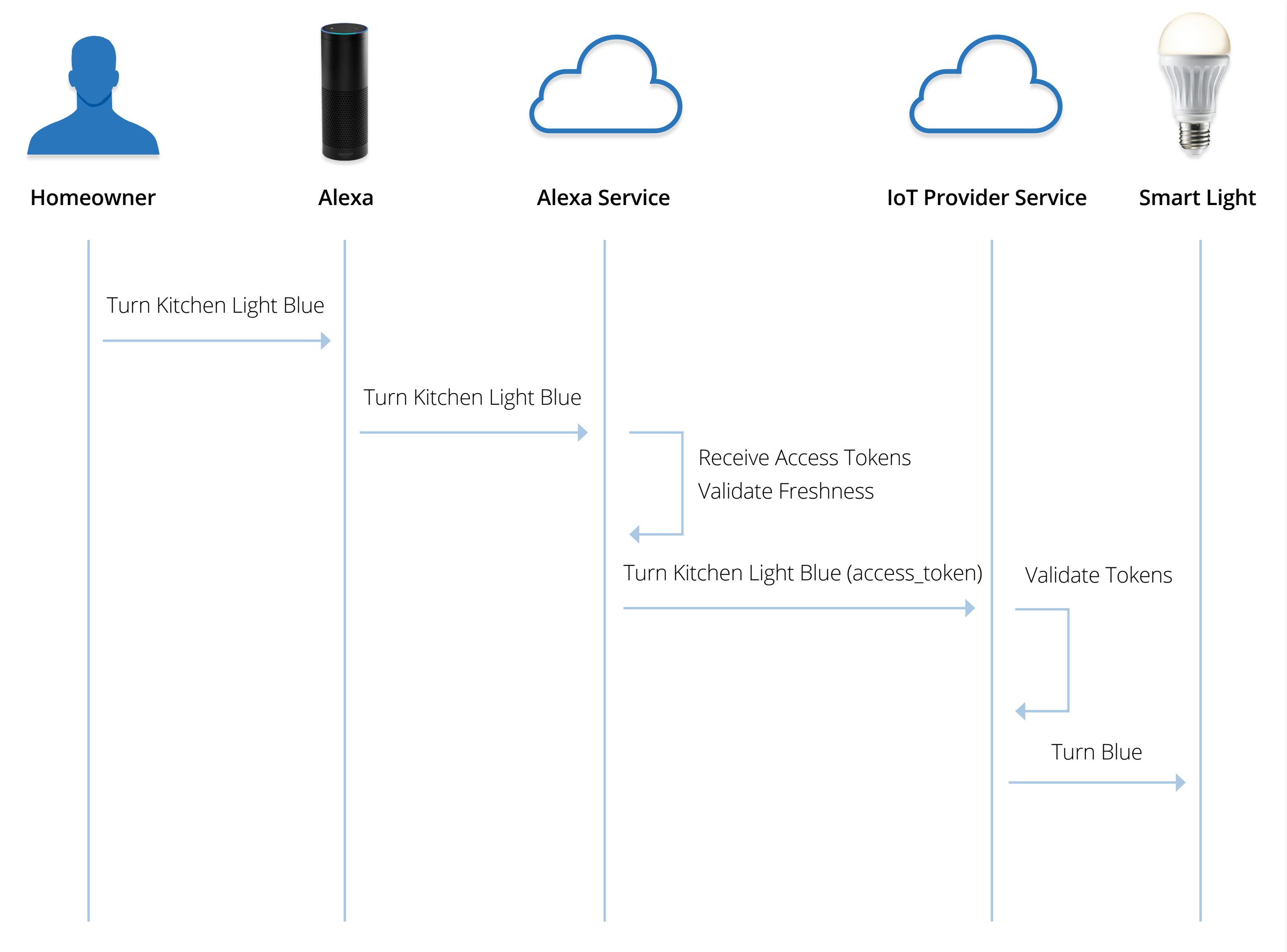
OAuth 2.0

OAuth 2.0 requires an authorization server that verifies user/device identity and then issues tokens for access. This is a centralized mechanism and requires that the environment the IoT product will operate within has access to an authorization server. This also means that the server must be protected from compromise.

OAuth 2.0 is now being used to support IoT access requirements. For example, an architecture may involve many diverse IoT devices that must interoperate. A smart hub or a service in the cloud might act as the authorization service for OAuth 2.0 transactions, where different IoT devices act as requestor and resource provider. This would all happen over TLS secured connections with schemes such as REST or MQTT.

If we look at Amazons' Alexa and the Smart Skills that can be created for that platform, we can see a good example. The Amazon Smart Home Skill API requires OAuth 2.0 authentication. In this example, the IoT developer of the service being supported provides the authorization server and the protected resource. The Alexa service (which hosts the smart skills) acts as the client. Figure 7 provides a view into the simplified workflow:

Figure 8



Users can create a linkage between a smart skill and an IoT app using their Alexa smart phone app paired with out of band visits to the IoT service. Read more about the Alexa smart skills authentication and authorization process **here**.

It is important to note that if a mobile application or thick-application is using OAuth 2.0, it is NOT recommended for authorization servers to perform client authentication using a secret (client_secret) distributed and hard coded within each application. Additionally, all tokens should be stored securely using the **iOS keychain** and **Android's Keystore** which gives developers the ability to utilize **secure hardware** on an Android device. The Internet-Draft "**Best Current Practice: OAuth 2.0 for Native Apps**" provides additional guidance for secure iOS and Android OAuth 2.0 implementations.

User Managed Access (UMA)

What about managing the complex policies associated with what apps and devices have access to each other? For this, we can turn to UMA for policy management. Where this service would physically exist is still up for debate, however it could be placed on a Wireless Access Point or in an enterprise gateway.

UMA is an OAuth-based access management protocol (**reference link**). The standard has been approved by the Kantara initiative. UMA supports the ability for parties to share and revoke information authorizations. This sharing can be accomplished on the fly. UMA seems to provide the most value in scenarios where a person is involved in the transaction between devices (e.g., through the concept of sharing in real-time, or more importantly being able to choose to share (or not) in real-time). UMA is a standard to consider when privacy is highly important and the sharing environment is complex.

11. Establish a Secure Key Management Capability

Key management is almost as arcane a topic as cryptography and is frequently implemented in an insecure manner, if implemented at all. Unauthorized disclosure of cryptographic keys renders the use of cryptography moot. Disclosure of many types of cryptographic variables can lead to catastrophic data loss even years after the cryptographic transaction has taken place. Key management therefore addresses how cryptographic keys are managed, and includes the following:

Table 13

Phase	Description
Key Generation	How, when, and on what devices keys are generated
Key Derivation	Constructing cryptographic keys from other keys and variables
Key Establishment Key Agreement Key Transport	Two party algorithmic computation of keying material Secure wrapping and sending of keys from one device to another
Key Storage	Secure storage of keys (frequently encrypted using 'key encryption keys') and in what type of device(s)
Key Lifetime	How long a key should be used before being destroyed (zeroized)
Key Zeroization	Secure destruction of key material
Accounting	Identifying, tracking and accounting for the generation, distribution and destruction of key material between entities

It is important to note that the technology and processes associated with commonly deployed Public Key Infrastructures are based on secure key management, however the IoT requires the industry to step back to appropriately implement the fundamentals of key management because not all IoT devices will interact with and consume PKI certificates. Many will be limited to consumption and use of symmetric encryption and authentication keys, for example.

Concerning the IoT, it is important to balance security versus performance. This comes into play regarding the lifetime established for cryptographic keys. In general, the shorter the lifetime the smaller the impact if a key is compromised; the shorter the lifetime, however, the more frequently keys either have to be generated or established and subject to accounting.

Upon expiration, new keys can be provisioned in myriad ways:

1. Sent by or retrieved from a central key management server using enterprise key management software.
2. Securely embedded in new software or firmware.
3. Generated by the device.
4. Established by the device with another party.

Secure key management also requires vendors to be very cognizant of the cryptographic key hierarchy, especially in the device manufacturing and distribution process. Built-in key material may emanate from the manufacturer (in which case, the manufacturer must be diligent about protecting these keys), overwritten and used or possibly discarded by an end user. Each key may be a prerequisite to evolving a device to a new state, or deploying it in the field as in a bootstrapping process. Manufacturers should document well the key management processes, procedures and systems used to securely deploy products. In addition, manufacturer keys need to be securely stored preferably in offline systems and Hardware Security Modules (HSM) within secure facilities.

Access controls to key management systems must be severely restricted given the large ramifications of the loss or tampering of even one single cryptographic key.

Table 14

Key Management

#	Question	Answer
1	Is secure storage for keys provided?	Secure storage implies keys are physically and (ideally) cryptographically protected
2	How are keys wiped after use or expiration?	Plaintext cryptographic key variables are known to exist in memory for long periods after use unless explicitly wiped. A good cryptographic library may zeroize keys under certain circumstances, but in others may leave it to the using application to invoke a zeroization service. It is important to ensure that security policies exist to zeroize key material under specific circumstances via both policy or technical controls. For example, TLS session keys should be immediately zeroized following termination of the session.
3	What key lengths are used?	In well-crafted algorithms, key length is frequently associated with resistance to cryptanalytic and brute forcing attacks. In general, it is strongly advised to adhere to best practices employed, for example, by the US government. In addition to sunsetting cryptographic algorithms, algorithm key lengths are also sunsetted. NIST Special publication 800-131 provides sound guidance on recommended algorithms and key lengths over time.
4	Is the source of entropy sufficiently random?	If an IoT device is generating any cryptographic keys or precursors, it is vital that the random number generator (or Random Bit Generator) be 1) a soundly designed RNG (e.g., ANSI X9.31 or NIST SP 800-90) and 2) seeded with an appropriate level of entropy from a good entropy source. Devices can use a variety of sources such as digitized random noise in electrical circuits, seemingly randomly timed events in operating systems, etc. When engineering an IoT device, perform min-entropy analysis on the seeding source of the device's random number generator.
5	How are certificates verified?	Ensure that certificates are either explicitly trusted or more likely, trusted through the loading of a root chain (chain of trust).
6	How are certificates revoked and expired?	It is critical to have the ability, like any typical PKI, to revoke certificates that have been associated with some type of misbehavior or device compromise. In addition, key lifetime in the case of digital certificates (X.509) is indicated in the credential. It is important that other devices that need to trust a given device are programmed or configured to check the expiration date of presented credentials and avoid trusting expired certificates. In addition, it is important to be cognizant and wary of cryptographic key (e.g., certificate) lifetimes that are too long and surpass the recommended lifetime of the underlying cryptographic algorithm or key length. For example, today, many valid SHA1-based certificates are still in effect when the SHA1 algorithm has underlying weaknesses and is being sunsetted.

Table 14

7	Who has access to key management systems? How?	Access to cryptographic key management systems must be severely restricted, physically and logically. Protected buildings and access-controlled rooms (or cages) are important for controlling physical access. Administrator and user access must also be carefully managed in terms of separation of duties (to application vs. host systems and services) and multi-person integrity (requiring more than one individual to invoke a sensitive service)
8	Are you using Perfect Forward Secrecy?	Perfect Forward Secrecy (PFS) is offered in many cryptographic key exchange and key establishment protocols. PFS is desirable because a compromise of one set of session keys will not compromise follow-on generated session keys. For example, in the case of Diffie-Hellman (DH) or Elliptic Curve Diffie Hellman (ECDH), a PFS scheme will generate ephemeral (one time use) DH/ECDH keys for each use such that the Shared Secret and its eventual derivation into cryptographic keys are always unique, session to session.
9	What key management protocols are you using?	This is a loaded question as there are relatively few key management protocols out there today - most tend to be embedded in proprietary offerings. Industry, in conjunction with the OASIS group, however, has created and begun adopting the Key Management Interoperability Protocol (KMIP) as a simple protocol for performing sender-receiver key management exchanges. It supports a number of cryptographic algorithms and was designed to answer the challenges of multi-vendor interoperability. KMIP has the ability to be leveraged in a variety of programming languages and used as a sub-protocol in any type of application or management layer protocol set.

Design Secure Bootstrap Functions

A secure bootstrap process serves as the foundation for many of the security features of an IoT device. The bootstrap process results in the provisioning of the initial credentials that will support authentication to various systems and with other devices.

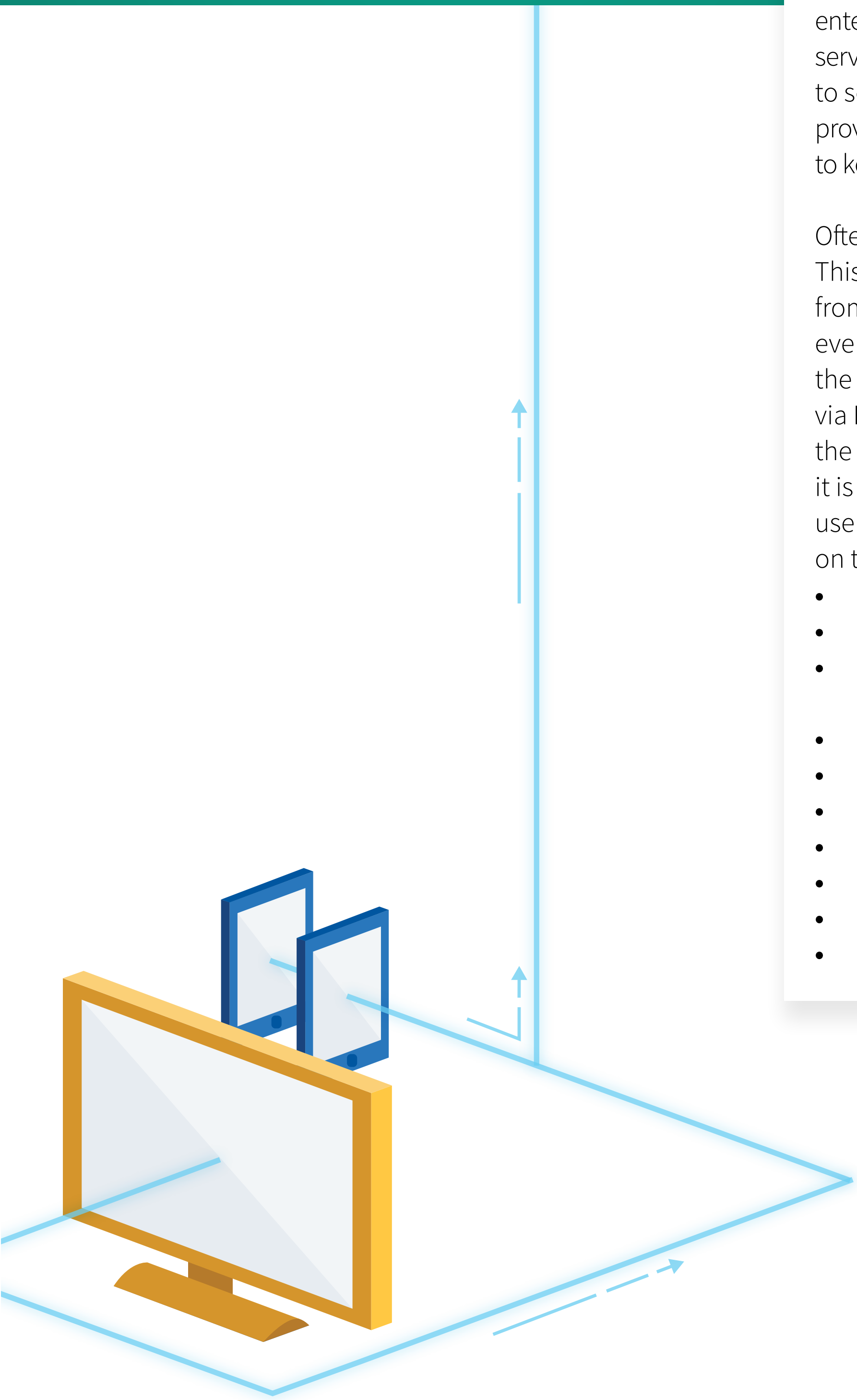
Oftentimes developers will pre-install a certificate into the IoT device, which allows the IoT device to securely bootstrap into a new system. For example, leveraging the installed manufacturer certificate to sign a certificate signing request (CSR) for an operational certificate. In these instances, it is often good practice to have additional verification procedures built-into the process, such as having an administrator vouch for the identity of the device during enrollment into the new PKI.

12. Provide Logging Mechanisms

One of the most critical aspects of securing an enterprise is understanding what events are taking place within that enterprise. Being able to see when users log into devices or services (and especially when they fail a login) are important to security practitioners. As developers of IoT products, provide your customer base with the information they need to keep an eye on their security posture.

Oftentimes IT assets use syslog as a logging mechanism. This allows administrators to easily access event data from devices and pass to powerful security information event management systems (SIEM) for correlation. Within the IoT, many IoT products provide some level of logging via REST APIs. Although there are multiple ways support the need for providing logging data to your customers, it is important that IoT products supporting enterprise users providing sufficient visibility into actions occurring on the device. This includes at a minimum:

- Connection Requests
- Authentications (failed / successful)
- Privilege abuse attempts / elevation of privilege attempts
- Receipt of malformed messages
- Successful / failed Firmware/software updates
- Local log-in attempts
- Configuration changes
- Account updates
- Protected memory access
- Physical tamper

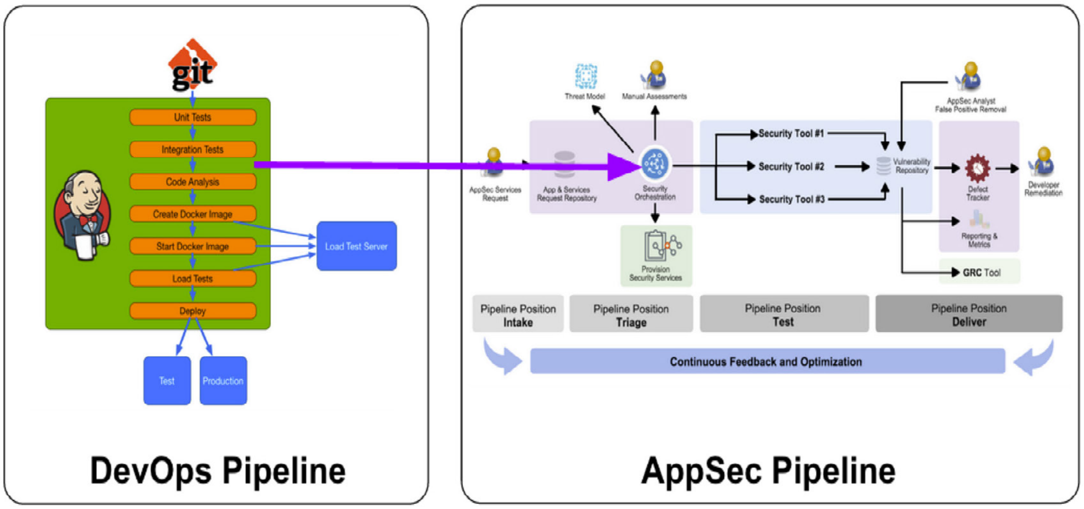


13. Perform Security Reviews (Internal and External)

The Open Web Application Security Project (OWASP) provides excellent tools to understand the software-based aspects of secure software development and the use of feedback loops that link design, development and test together. Remember within the IoT however that software is only one portion of the effort (includes hardware that must also be reviewed). Approaches such as the Common Criteria can be used as a reference for building in hardware based security features within your product.

Feedback loops allow for design updates that are driven by the identification of weaknesses during test or fielding. Figure 8 provides OWASP's view into the AppSec pipeline.

Figure 9



OWASP APPSec Pipeline

OWASP's AppSec Pipeline shows the need for continuous feedback and optimization across an IoT product's lifecycle. Defects /vulnerabilities that are identified must be fed back into the design and threat modeling process, resulting in updates to hardware and software baselines that then must be re-tested to ensure that the patches did not introduce new vulnerabilities.   The IoT device security testing process is designed to discover these vulnerabilities.

There are many types of tests that can be employed for IoT device developments and each plays a critical role in maintaining the security posture of the device:

1. Static Application Security Testing (SAST)
2. Dynamic Application Security Testing (DAST)
3. Interactive Application Security Testing (IAST)
4. Attack Surface and Vectors
5. 3rd Party Library
6. Fuzzing
7. Customized per threat vector

IoT product developers should also leverage a 3rd-party evaluation partner to perform a security evaluation of your IoT device. There are services available for a fee, as well as pro-bono services available that will accept your hardware and use volunteer researchers to test. Some companies have their own internal evaluation teams that can be leveraged for this analysis. Whichever approach is chosen, the feedback from this evaluation should provide valuable insight into the security posture of the device and allow for an identification of required mitigations. Examples of industry organizations that offer 3rd-party evaluations include builditsecure.ly, and crowdstrike.

ICSA Labs has also developed an assurance program for testing IoT devices. ICSA focuses on testing of six components:

- Communications
- Alert/logging
- Platform Security
- Cryptography
- Physical Security
- Authentication

Review the ICSA Labs offering **here**.

Underwriters Laboratory (UL) has also created a new assessment service for software and security that is geared towards IoT devices. Review the UL assessment offering **here**.

Appendix A –
Categorizing IoT Devices

In order to aid in identifying unique threats for your IoT device developments, this section categorizes various types of IoT devices, provides examples of the typical environments that these devices are deployed within, and then outlines typical threats for each implementation. This section breaks out examples by the following types of IoT devices:

1. Consumer
2. Health
3. Industrial
4. Smart Cities

Table 15 provides definitions for the various threat actors discussed herein.

Table 15

Threat Actor	Description
Nation-State	Enemy state involved (directly/indirectly involved) in security incidents motivated by financial gains, access to intellectual property, to gain political mileage or to inflict damage to critical Information Systems.
Cyber Terrorist	Carry out an attack designed to cause alarm or panic with ideological or political goals. Generally these threat actors are part of a known terrorist organization.
Hacktivist	One who performs attacks in order to draw attention to a political cause such as free speech or human rights or hinder the support of a cause. They are politically motivated.
Hacker	A person who uses computers to gain unauthorized access to data.
Organized Crime	These are groups of criminals that intend to engage in illegal activity, their activities are driven by monetary greed. Attacks are designed to either extort money from subjects, or the actors commercially funded to carry out such attacks.
Individuals	A specific person or group acting on their own, and not affiliated with any group or association. Does not fall under any other category.
Prankster	
Insider/System User	Authorized user, using his/her credentials to access unauthorized data.
Thief	

Consumer IoT Devices

Consumer facing IoT devices have received significant attention from the security community, given their low-cost of acquisition/ high availability. Although this class of IoT devices may not seem to impact enterprise security postures, a recent **report from OpenDNS** has shown that these types of devices are being used within corporate environments and are often placed on the enterprise network. Maintaining a sound security posture for these types of devices benefits everyone.

Table 16 provides a listing of different types of consumer IoT devices and the environments that they may operate within.

Table 16

IoT-Device Type	Examples	Typical Enviornments	Typical Threats
Wear-ables	Fitness/lifestyle/ fertility trackers/ smart watches	Indoor & Outdoor Locations Healthcare Gym Office	• Individual attempting to gain access for bragging rights • Dangerous medicine intake/ lifestyle changes due to misreporting. • Full address book information and third party smartwatch application data. • Location information vulnerabilities may give access to stalkers.
Smart Home	Lighting, thermostat, door locks, garage door openers, pool pumps, water sprinkler, Refrigerator, vacuum cleaner, TV, dishwasher, washing machine, music system, baby monitors, video monitors, audio listening devices, door bells	Home Indoor Hotels Restaurants Offices Hospitals/Healthcare facilities Baby care centers,	• Prankster intent on causing concern/confusion • Individual attempting to gain access for bragging rights • Hacker attempting to use monitor to jump onto network • Financial; consuming utilities through IoT device. Operating devices to the point they fail. • Thief attempting to identify patterns/ person availability by tripping the alarm.

As can be seen from the above table, the typical threats faced by consumer IoT devices are those of opportunity, however with the increased connectivity within both the home and corporate environments - it is likely that we will see targeted attacks focused on penetrating a home network in order to gather knowledge useful for compromising an enterprise. It is also likely that criminal elements will begin to target weaknesses found in consumer IoT implementations as a way to gather information that will support various financial crimes. Given this, it is important that manufacturers of consumer IoT devices take the security posture of their products seriously and use innovative methods to balance the challenge of making their products usable and secure.

Smart Health Devices

One of the industries that stands to benefit the most from the IoT is health care. Medical instrumentation is being updated to be wirelessly connected to computers in the office and the cloud. Pharmacological companies are IoT-enabling pill bottles and researchers are even busy developing implantable diagnostic tools. Given the functionality provided by smart health devices, developing these products securely is critical to safeguarding patients from harm.

Table 17 provides a listing of different types of consumer IoT devices and the environments that they may operate within.

Table 17

IoT-Device Type	Examples	Typical Enviornments	Typical Threats
Connected Medical Equipment	Infusion pumps, glucose monitors, baby monitors	Home Hospital Outpatient Facilities	• Organized Crime attempting to cause harm to patients. • Medical information disclosure.
Connected Self Diagnosing Devices	Diabeto, Azoi	Home Outpatient Facilities	• Unreliable diagnosing due to device's software or hardware failure. • Organized Crime attempting to cause harm to VIP patients.
Connected Toold	Wireless stethoscope, thermometer	Home Hospital Outpatient Facilities	• Unreliable diagnosing due to device's software or hardware failure.
Implantables	Pacemakers, ingestible cameras,	Body	• Life violation. • Assault of human life by indicating fake information to devices and making them take harmful decisions.
Ancillary Products	Smart Pill Boxes	Home Pharmacy	• Unknown side effect to human body .

Smart health devices must often comply with various legal regulations. In the United States for instance, devices and many device updates must be reviewed by the FDA prior to being sold. The following guidance should be reviewed by organizations developing smart health products:

- **Cybersecurity for Medical Devices and Hospital Networks: FDA Safety Communication**
- **Medical Devices: Digital Health**
- **ISO14971 Application of Risk Management for Medical Devices**

Industrial IoT Devices

Industrial use cases have been around for a while to help automated data transmission and measurement between mechanical or electronic devices. These continue to play a vital role in the industrial segment. In the name of better production and more safety, various wearables and industrial beacons are being incorporated in factories.

Table 18 provides a few such installations and their typical threats.

Table 18

IoT-Device Type	Examples	Typical Enviornments	Typical Threats
SCADA Systems	manufacturing, water, electric, gas, nuclear, PLCs, RTUs,	Factory (physical premises)	• Cyber or Eco terrorist attempting to cause physical destruction or harm. • Prankster attempting to disrupt operations. • Hacktivist attempting to disrupt operations. • Nation-state attempting to cause mass destruction. • Insider attempting to “get back” at employer. • Supply chain attacks relevant to purchased components.
Industrial-use Unmanned Aerial Systems (UAS)	Small UAS, delivery platforms, monitoring and surveillance	Remote and difficult to reach locations	• Thief attempting to steal delivery or drone itself • Cyber terrorist attempting to cause disruption or harm (direct platform into crowd or active scenario with fixed wing fire-fighting) • Prankster attempting to take control of device for fun
Mining	Autonomous drilling rig, autonomous hauling truck, pressure sensors, proximity sensors, thermometers, motors, pumps, gyroscopes	Factory (physical premises)	• Cyber terrorist attempting to cause physical destruction or harm • Hacktivist attempting to disrupt operations • Insider attempting to “get back” at employer
Manufacturing	Robotics		• Cyber terrorist attempting to cause physical destruction or harm • Hacktivist attempting to disrupt operations • Insider attempting to “get back” at employer

The following guidance should be reviewed by organizations developing ICS-based (Industrial Control System) IoT products.

- **Recommended Practice: Improving Industrial Control Systems Cybersecurity with Defense-In-Depth Strategies**
- **Guide to Industrial Control Systems (ICS) Security**

Smart Cities

We categorize public facing IoT systems and devices as supporting many different capabilities throughout a smart community.

Table 19

IoT-Device Type	Examples	Typical Enviornments	Typical Threats
Smart City Infrastructure and Services	Traffic lights, environmental sensors, digital billboards, smart communications (smart antennas), elderly monitoring, smart hospitality (hotel room locks), emergency management, environmental monitoring	Physically exposed, geographically dispersed	• Cyber terrorist attempting to cause physical destruction or harm. • Hacktivist attempting to disrupt operations. • Insider attempting to "get back" at employer.
Smart Transportation	Connected Vehicles, autonomous vehicles, road side equipment, transportation centers, pedestrian warning devices,	Physically exposed, geographically dispersed, sometimes moving	• Cyber terrorist attempting to cause physical destruction or harm. • Hacktivist attempting to disrupt operations. • Insider attempting to "get back" at employer.

Appendix B – References

[1] http://www.securityweek.com/serious-security-flaws-found-hospira-lifecare-drug-pumps

[2] http://www.devttys0.com/2012/11/reverse-engineering-serial-ports/

[3] Y.2060. Overview of the Internet of things. International Telecommunications Union (ITU-T), 6/2012. Available at https://www.itu.int/rec/T-REC-Y.2060-201206-I

[4] https://www.rapid7.com/docs/Hacking-IoT-A-Case-Study-on-Baby-Monitor-Exposures-and-Vulnerabilities.pdf

http://devops.com/2014/10/13/devops-iot/

http://www.itproportal.com/2015/05/05/what-internet-of-things-means-for-devops/

http://theagileadmin.com/what-is-devops/

http://techcrunch.com/2015/05/15/what-is-devops/#.2rzef7r:5XYN

[10] http://www.informationweek.com/mobile/mobile-applications/11-iot-programming-languages-worth-knowing/d/d-id/1319375?image_number=1

http://www.emdt.co.uk/daily-buzz/how-deal-it-security-threats-connected-medical-devices

http://www.adlawbyrequest.com/wp-content/uploads/sites/491/2015/01/IOT-Report-Lah-1.29.15.pdf

[20] http://www.ti.com/lit/wp/slay041/slay041.pdf

[21] http://micrium.com/iot/iot-rtos/

[22] H. Ning and H. Liu, “Cyber-Physical-Social Based Security Architecture for Future Internet of Things”, Advances in Internet of Things, vol. 02, no. 01, pp. 1-7, 2012.

[23] PubNub, “A New Approach to IoT Security”, 2015.

[24] Wind River, “SECURITY IN THE INTERNET OF THINGS”, 2015.

Appendix C – IoT Standards and Guidance Organizations

Table 20

Organization	Description
AIOTI	The Alliance for Internet of Things Innovation (AIOTI) was launched by the European Commission to support the development of a European IoT ecosystem, including standardization policies. https://ec.europa.eu/digital-agenda/en/alliance-internet-things-innovation-aioti
AllSeen Alliance	A 180-member industry group, the AllSeen Alliance promotes widespread adoption of an interoperable peer communications framework based on AllJoyn for devices and applications in IoT. https://allseenalliance.org/
Cloud Secuity Alliance (CSA)	Internet of Things Workgroup extends and builds on the spec from ITU-T Y.2060 and looks at the aspects interfacing between cloud based systems and edge devices covered in ITU and other standards. https://cloudsecurityalliance.org/group/internet-of-things/
ETSI	ETSI's Connecting Things effort is developing standards for data security, data management, data transport and data processing related to potentially connecting billions of smart objects into a communications network. http://www.etsi.org/technologies-clusters/clusters/connecting-things
IEEE (including P2413)	The IEEE has a dedicated IoT initiative and clearinghouse of information for the technical community involved in research, implementation, application and usage of IoT technologies. http://iot.ieee.org/
IERC	The European Research Cluster on the Internet of Things coordinates ongoing activities in the area of IoT across Europe. http://www.internet-of-things-research.eu/
Internet Engineering Task Force (IETF)	The Internet's premier standards setting body has an IoT Directorate that is coordinating related efforts across its working groups, reviewing specifications for consistency, and monitoring IoT-related activities in other standards groups. https://trac.tools.ietf.org/area/int/trac/wiki/IOTDirWiki
IIC	The Industrial Internet Consortium (IIC) has teamed up with the OIC to accelerate the delivery of an industrial grade IoT architectural framework. IIC released a reference architecture for IoT in 2015. http://www.industrialinternetconsortium.org/
Internet Governance	IGF sponsors the Dynamic Coalition on IoT, which hosts open meetings to discuss global challenges that need to be addressed regarding IoT deployment.
Forum	http://www.intgovforum.org/cms/component/content/article?id=1217:dynamic-coalition-on-the-internet-ofthings
Internet of Things Consortium	This industry group provides consumer research and market education aimed at driving adoption of IoT products and services. http://iofthings.org/#home

Table 20

IoT Security Foundation (IoTSF)	A group purely dedicated to building standards for IoT Security https://iotsecurityfoundation.org/
IP for Smart Objects (IPSO) Alliance	Dedicated to enabling IoT, IPSO seeks to establish IP as the basis for connecting smart objects through education, research and promotion. http://www.ipso-alliance.org/
ISO/IECJTC-1/SC 27	ISO issued a preliminary report on IoT in 2014 as well as a Smart Cities report. The group has ongoing subcommittees in both areas. (ISO/IEC JTC 1/SC 27 10) http://isotc.iso.org/livelink/livelink/open/jtc1sc27 http://www.iso.org/iso/internet_of_things_report-jtc1.pdf
ISOC's Internet of Food SIG	This special interest group leads discussion on the technical infrastructure standards needed for the food industry in the future. http://internet-of-food.org/
ITU	The ITU hosted an IoT Global Standards Initiative, which concluded its activities in July 2015, followed by the formation of a new Study Group 20 focused on IoT applications. http://www.itu.int/en/ITUT/studygroups/2013-2016/20/Pages/default.aspx
MAPI Foundation	The Manufacturers Alliance for Productivity and Innovation (MAPI) s developing Industrie 4.0 for industrial applications of IoT. https://www.mapi.net/research/publications/industrie-4-0-vsindustrial-internet
OASIS	OASIS is developing open protocols to ensure interoperability for IoT. The group chose Message Queuing Telemetry Transport (MQTT) as its messaging protocol of choice for IoT and has optimized MQTTS-N for wireless sensor networks. OASIS has three technical committees in IoT overseeing MQTT and two other standards, Advanced Message Queuing Protocol (AMQP) and OASIS Open Building Information Exchange (oBIX). https://www.oasis-open.org/committees/tc_cat.php?cat=iot
oneM2M	Dedicated to developing machine-to-machine communications architecture and standards, this multi-vendor group is focused on telemedicine, industrial automation, and home automation. Its goal is a common M2M Service Layer that can be embedded in hardware and software. http://www.onem2m.org/
Online Trust Alliance	This group of security vendors has developed a draft trust framework for IoT applications, focused on security, privacy, and sustainability. https://otalliance.org/initiatives/internet-things
Open Interconnection Consortium	OIC is defining a common communication framework based on industry standards to wirelessly connect and manage the flow of information among IoT devices. It sponsors the IoTivity Project, an open source software framework for device-to-device connectivity. http://openinterconnect.org/
The Open Management Group	This technical standards consortium is developing several IoT standards, including Data Distribution Service (DDS) and Interaction Flow Modeling Language (IFML) along with dependability frameworks, threat modeling, and a unified component model for real-time and embedded systems. http://www.omg.org/hot-topics/iot-standards.htm
Open Web Application Security Project	OWASP sponsors an IoT Top Ten Project, which is designed to help manufacturers, developers, and consumers understand related security issues with its list of the most significant attack surface areas for IoT. https://www.owasp.org/index.php/OWASP_Internet_of_Things_Top_Ten_Project
Smart Grid Interoperability Panel	SGIP has an effort called EnergyIoT focused on new opportunities for IoT within the energy industry. The group's OpenFMB is a utility-led project that is incorporating common utility data models and IoT communication protocols to create an Open Field Message Bus. http://sgip.org/focus-resilience
Thread Group	This group of smart home vendors is developing a common networking protocol that will support IP-enabled devices in the home such as appliances, lighting, and security systems. http://threadgroup.org/About.aspx

Table 20

W3C Web of Things (WoT)	a EU FP7 supported project to identify use cases and requirements for open markets of applications and services based upon the role of Web technologies for a combination of the Internet of Things (IoT) with the Web of data. https://www.w3.org/WoT/
Securing Smart Cities	Securing Smart Cities is a not-for-profit global initiative that aims to solve the existing and future cybersecurity problems of smart cities through collaboration between companies, governments, media outlets, other not-for-profit initiatives and individuals across the world.
UL2900	The UL Cybersecurity Assurance Program (CAP) brings peace of mind. CAP certification verifies that a product offers a reasonable level of protection against threats that may result in unintended or unauthorized access, change or disruption. http://industries.ul.com/software-and-security/product-security-services/product-testing-and-validation
NIST CPS Public Working Group (CPS PWG)	The Cyber-Physical Systems Public Working Group (CPS PWG) will bring together experts to help define and shape key aspects of CPS to accelerate its development and implementation within multiple sectors of our economy. - http://www.cpspwg.org/Portals/3/docs/CPS%20PWG%20Draft%20Framework%20for%20Cyber-Physical%20Systems%20Release%200.8%20September%202015.pdf

Appendix D – Other Guidance Documents

There are many other guidance documents that would be useful for IoT product designers and developers to examine.

- The PRPL Foundation released **Security Guidance for Critical Areas of Embedded Computing** in 2015. This guidance provides details on concepts such as using a security separation approach and enforcing secure development and testing for embedded systems.
- The Defense Information Systems Agency (DISA) has a long history of providing Security Technical Implementation Guides (STIGs) for development teams to use in securing their products. One in particular that IoT product developers may find useful is the **Application Security and Development STIG**. This STIG provides guidance on development, design, testing, maintenance, configuration management and training.
- Microsoft has spent many years defining a **Secure Development Lifecycle**. The Microsoft SDL outlines a series of practice areas that include the creation of quality gates and bug bars, establishing design requirements and performing an attack surface review.
- The OWASP IoT Project is doing great work in defining the threats to the IoT. They have also published a **testing guide for IoT devices** that details how to test for things such as insecure mobile interfaces, insecure software/firmware and poor physical security.