CTA Technical Report

Securing Connected Devices for Consumers in the Home

CTA-TR-12

(Formerly CEA-TR-12)

November 2015

Consumer Technology Association

Consumer Technology Association

(Formulated under the cognizance of the CTA **R7 Home Networks Committee**.)

Published by
©CONSUMER TECHNOLOGY ASSOCIATION 2015
Technology & Standards Department
www.CTA.tech

All rights reserved

FOREWORD

This technical report was developed under the auspices of the Consumer Electronics Association's R7 Consumer Electronics Networking Committee.

Table of Contents

1. Introduction

1.1. Mission

This CEA technical report addresses security concerns related to the connected devices for consumers in the home. Many of these devices fall into what is being referred to as the smart home or the Internet of Things (IoT). There is a general lack of consensus on the definition of the IoT. For the purpose of this report we will be addressing the collection of uniquely identifiable, accessible and consumer oriented Internet-connected electronic devices and the services that drive their functionality.

This report will use an example product with wired and wireless connections.[1] Specific technologies may be called out in order to discuss relevant security properties, but the intent of this report is to introduce security principles that are technology agnostic.

No technical report could fully address all connected device security concerns. It is the goal of this report to provide a solid foundation for building security into the development process. This will include a discussion of Security Maturity Models and an examination of some common connected device vulnerabilities and mitigations.

1.2. Audience

This technical report is aimed at a variety of professionals involved in product development lifecycles. Section 2. Process and the Maturity Model will be of particular interest to product managers, software and hardware architects, software engineers, and technical project managers. Section 3. Vulnerabilities and Mitigations will be of particular interest to software and hardware architects, and software engineers. No formal background in security is assumed by the authors of this report.

1.3. Security and Policy

What does it mean to be secure, when building technology? What does it mean to operate securely? From the user's point of view, security is technology that protects the user from malicious activity of others. But, who are the others and what is malicious activity?

This last question is not about technology. "Security" is a combination of *policy*, and *technology to enforce that policy*. A policy might be that "no strangers are allowed in the house without permission"; the technology might be a door lock.

Different parties inevitably have different wants or needs, and this means they have differing security policies. A company may consider it to be a legitimate practice to sell personally identifiable information (PII) to third parties; that position is the company's policy. The consumer may consider their PII private; this is a different policy. As a result of these different interests and policies, security is not a one-size-fits-all concept.

Security policies often conflict between interested parties: consumers, manufacturers, responsible government agencies, infrastructure suppliers, and even malicious 3rd party actors and nation-states.

[1] The *MeshPet 2000* hypothetical product, explained in a later section, includes two wireless links. Every wireless protocol has its own security pros and cons, but the discussion in this report is intended to be technologically agnostic. As a result, the recommendations apply broadly to various connectivity solutions such as Ethernet, Multimedia over Coaxial (MoCA), Bluetooth LE (Low Energy), IEEE 802.11 (Wi-Fi), cellular wireless, ZigBee, Zwave, RFID or near field communications (NFC).

These security policy differences lead to security policy breaches, the potential of which are called threats. An important concept in security is that *the threat environment dictates what must be secured and to what level*. Determining the different ways in which one party should be protected from another will dictate the boundaries and security properties that must be enforced.

1.4. Security Guarantees

Proving a non-trivial system secure is unlikely—how can one ensure that all possible threats will fail? The concept of provable security, using mathematical proofs to reduce a complex system to an underlying "hard" problem continues to be controversial[2]. Proving something is insecure can frequently be simpler than proving it is secure, but it is not deterministic. That is, there is no simple set of procedures to follow and know for certain that an application is insecure.

Secure?

"Beware of bugs in the above code; I have merely proven it correct, not tried it."

-Donald Knuth

If the list of all possible insecurities could be produced, and they could all be checked, this proof by exhaustion would prove a program secure. Though it is widely accepted that no actual conclusive proof can be constructed in this fashion, this approach is commonly the primary motivation in much security research and security analysis. Understanding and testing vulnerabilities becomes a tractable aspect of the problem that can be done effectively and scaled to large scale software creation processes, even if it is a heuristic and does not lead to any direct proof of security or insecurity.

Given that proving security may be impossible, and proving insecurity is not deterministic, what can be done? Is it possible that software can be written in such a way to improve the odds that a pragmatic security review has a better chance of identifying insecurities?

A threat modeling analysis might review the data flow within a given block of code, to determine where data comes from and where it goes. Enumerated threats could be vetted against the code to determine, in general terms, the security posture of the application. We can prove that, in the limited case, an application may not be vulnerable to specific expressions of vulnerability classes. In doing so it is possible to gain confidence in the security properties of the resulting software.

1.5. Process and Maturity Model Overview

The industry standard practice for producing product with some level of security hardening is to treat it like a process. This is a similar path as used to ensure product quality in manufacturing. Manufacturing companies have defined processes for supplier quality approval and ongoing quality assurance; for designing for quality; for manufacturing. Over the past few decades these quality processes have been proven effective.

Section 2 of this report addresses the security process and maturity models. A coherent security process can allow organizations to build security into products and services from the beginning. This section will discuss the security process in terms of one industry-accepted security maturity model (BSIMM). The model is built from four "pillars": Governance practices will be discussed as they relate to engaging everyone in building and deploying secure technology. Intelligence practices will be described that look at the deeper perspective of organizing security into a product under development. The secure development lifecycle and its interaction with maturity models zeros in on the product development process, and

[2] http://www.ams.org/notices/200708/tx070800972p.pdf

correctly integrating security tasks into the overall process. Finally an important discussion of how to plan for the customer use and adaptation of technology will address the deployment phase of a product.

No process or product is perfect, and nothing is ever completely secure. Through the use of maturity models and secure development lifecycles it is possible to produce safer and more secure products and services.

Each major subsection includes a portion titled, "*Where to Start*", for concrete guidance on adopting these BSIMM pillars in an organization.

1.6. Vulnerabilities and Mitigations Overview

Section 3 of this report addresses the specific architecture and implementation flaws that may have security impact in a product or service. Using a hypothetical product, called the *MeshPet* 2000, specific classes of vulnerabilities will be discussed. In each case there will be the description of the vulnerability, an examination of its impact and a brief reference to known mitigation strategies.

The *MeshPet* 2000 is the next generation of pet monitoring solutions. It incorporates embedded computing concepts, leverages multiple wireless interfaces and communicates with both mobile devices and cloud services. This section will describe network vulnerabilities and mitigations, host vulnerabilities and mitigations, as well as application threats and mitigations.

2. Process and the Maturity Model

Why a maturity model? How is this like Kaizen?

One answer is that this is how software security experts approach the problem, so to learn how to secure a product is to learn how to implement a maturity model. There is logic behind this conventional wisdom, of course.

For the Internet of Things, the "delivered product" may include an embedded device, wireless links, cloud data storage, a web-based UI, and other elements—a chain of many potentially weak links. Securing only one element is like hardening only one link in a chain.

Worse, every product is different, and every business has its own characteristics. Without a repeatable process, the security of a specific product requires solving many problems that will need to be solved again for the next product.

This challenge is the same as the challenge of product quality. Companies do not deliver quality products by heroic individual efforts. They deliver quality by ongoing improvement of institutional-level tactics. The Toyota Production System[3] is a well-known example of a process to produce the right results: A quality product in an efficient manner.

The security maturity model replaces the "craftsman" approach to locking down a system, just as *Kaizen* and quality process improvements replaced "test and reject" methods.

The security process is a set of tasks and steps incorporated into a development and engineering process to build security in from the beginning. Ultimately, the desired output is to have a cost-effective engineering effort that yields a resilient hardened product. By structuring the necessary activities as a process, engineers can be consistent about executing those activities. This section will discuss the security process in terms of the industry-accepted *security maturity model*.

A security maturity model incorporates tasks to understand the security properties of a given design and implementation, and identify bugs and necessary trade-offs as early as possible. The earlier issues are identified, the cheaper they are to fix. Integrating a security maturity model into an organization's practices and culture are the industry's current best solution to identify and resolve security issues as early in the process as possible.

There are three main security maturity models in this area. Although each maturity model approaches the goal from a particular point of view it is not necessary to be dogmatic about the choice. They each have similar goals, institute similar steps and sub-processes, and ultimately work in similar ways. The models can be mapped from one to the next with relative uniformity and there are few gaps not addressed by one of the models. It is largely up to organization adopting the maturity model to choose one that most closely mimics how they think about their own process, and what nomenclature they use.

2.1. Introduction to Security Maturity Models

The three main security maturity models being used by industry come from a number of different sources:

[3] https://en.wikipedia.org/wiki/Kaizen#Implementation

- Microsoft created the **SDL**[4], originally for internal use to address consistent and frequent defects in their software and was eventually codified and evangelized to the industry.
- The OWASP[5] organization created **OpenSAMM**[6] based on their view of the necessity of a security process as a key requirement for creating secure web applications.
- Cigital and Fortify jointly developed **BSIMM**[7], with the assistance of many of their customers.

The BSIMM model will be the focus of this report.

Each maturity model has the high level goal of framing a process that can be used to engineer resilient, secure software. The differences between them vary, but much of the difference is the perspective from which it is written and some of the language used to describe the tasks and sub-processes.

2.2. BSIMM

Unlike the other two, BSIMM was not created by a single organization but as the product of a study of a number of major technology organizations. These organizations had each implemented numerous independent processes to attempt to improve their security stance. BSIMM’s longitudinal study is ongoing, and new versions of BSIMM have been released to incorporate additional information from the original study participants as well as new ones that have entered the study.

Although the way BSIMM came to be is significantly different, the corresponding process contains no real surprises. In fact, some of the similarities are natural given that many of the participants in the study have implemented the SDL or OpenSAMM methodologies. So it is no surprise that some of the steps and tasks have filtered through to the BSIMM recommendation. Given the nature of BSIMM, it is by definition a collection of industry practices.

For organizations that would like to follow industry best practices the BSIMM model offers a unique perspective on what a few members of the industry are doing and a collected set of practices that are applied by study participants. Evaluating the model objectively, it provides a pragmatic set of guidance that can be put to use to improve the security of a construction process and reduce the number of vulnerabilities and identified security issues once technology products have been created. One important difference using the BSIMM approach is that it will by definition *eventually* incorporate new strategies and processes that enter into the industry and are being adopted. It is unclear yet if other efforts will adapt quickly to such changes or not. To pick a specific example, the SDL might be revised faster than BSIMM to recommend a new technique, but BSIMM is guaranteed to pick it up as it enters into the organizations participating in the study while the OpenSAMM may choose for some reason to not incorporate it.

Regardless of what methodology is chosen, it is clear that incorporating a set of security practices into the development process improves the resiliency of the outputted product. There are minor differences between an organization practicing these different variations, but an overwhelming difference between those organizations and one without any security process at all. There are some fledgling or niche models not described here, such as CMM-SSE and CLASP; and it is likely that as time marches forward the

[4] http://www.microsoft.com/security/sdl
[5] https://www.owasp.org
[6] http://www.opensamm.org
[7] http://www.bsimm.com

industry will continue to innovate new approaches. No matter which methodology is used, one should be used to create a process capable of exerting a downward trend on security issues identified in released technology products.

Because the BSIMM study samples a broad variety of technical companies, and because some of the nomenclature may better apply to hardware manufacturers who also do software development, BSIMM was chosen as the example methodology to include in this technical report. BSIMM is also a little more flexible in its á la carte approach. As noted before, the distinctions to be drawn are small and the other models could form a similar backbone on which to structure an organizational process.

For companies just getting started in adopting a security process, the material in the following section is good to get started and compatible with *connected device* product development. The rest of Section 2 details the four pillars of BSIMM, and the different practices in each pillar. The goal is to give a high level understanding of what BSIMM process looks like with an eye towards how a company looking to adopt a secure development process could adapt the highest value activities to start to get a security benefit right away.

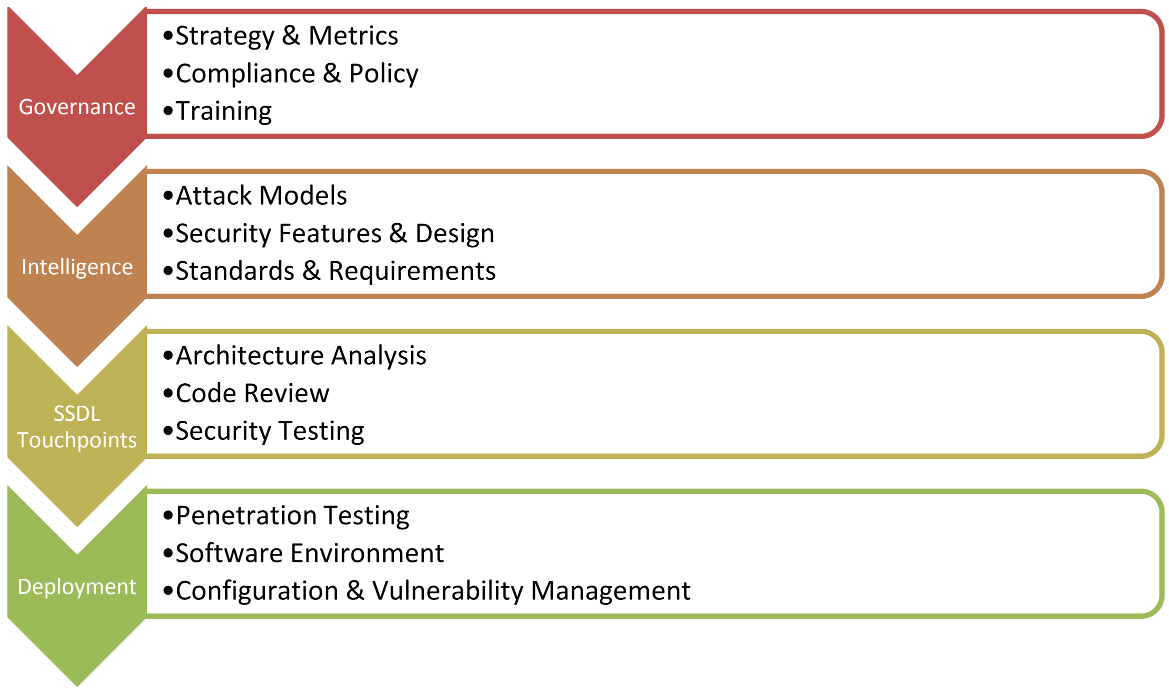


Figure 1 BSIMM at a glance

2.2.1. Governance

The first pillar of BSIMM surrounds the preparatory work an organization must do to have a sufficient foundation on which to build a resilient secure development process. Governance practices focus on dealing with the current position of the company, within its industry including regulatory and compliance regimes. But more than just dealing with compliance, it supports a set of internal changes that can set the stage for everyone involved in building and deploying secure technology to be ready to embrace security.

Strategy and Metrics

Strategy and metrics focuses on developing an over-arching strategy and a context in which to consider secure development. Practices like publishing processes, internal evangelizing of security and publishing the current state of security at the company can help build understanding and provide some awareness internally. Further, educating executives about the importance of security and the potential risks from consequential, direct and reputational damage can make decision makers aware of the need for security. Beyond that, strategy includes identifying the existing gates or process that will incorporate security tasks and signoff. Strategy also includes creating a central security team and potentially satellite security team members distributed throughout the organization to better drive awareness. Metrics are the last point, and good metrics can help provide insight on what is working and what is not. Further, simple metrics can be powerful in driving budget considerations by properly informing the business on the pros and cons. Tracking all of this information across the organization gives a view and capability to understand the businesses technical risk at a glance, aiding key decision making from the macro, business-wide scale, to individual, product or team decisions. The NIST Internal Report 7564[8], Directions in Security Metrics Research, provides an excellent overview of the state of security metrics and their various properties.

Compliance and Policy

Compliance and regulatory compliance can be a significant consideration for organizations subject to existing government or industry regulation. For such companies, it is necessary to ensure that regulations are followed and that this can be demonstrated to auditors to avoid penalties or sanctions. Even for companies not subject to strict regulatory regimes, strong security policies can ensure that the right security decisions are made. When considering compliance, one of the first things to do is to combine different compliance regimes together so that a complete set of regulatory requirements can be understood as they apply to the software and hardware creation. Later, decisions about regulations like tracking regulatory information and justifications for decisions, as well as providing for a sign-off process can be added to make managing regulatory constraints and the auditing process simpler.

Beyond pure regulation though, policy management applies to companies that have significant regulatory constraints as well as those who are unencumbered. Policy can be used to as a framework to deal with regulation, but can also apply to non-regulatory requirements of the development process.

Training

Training is one of the most important practices to begin with, particularly for organizations that are beginning with security. Security awareness training is frequently the first training that is created, to help everyone that is involved with development, from engineers to executives, to have a solid understanding of the importance of security. More advanced trainings on a variety of specific topics can then be used and custom tailored to different target audiences like business owners, developers and quality assurance engineers. Advanced topics frequently include training on tools, technology stacks or specific kinds of security defects.

Where to Start

Governance is the BSIMM pillar that prepares an organization to adopt and accept a security process. The focus is around understanding the requirements of the business, planning a process and understanding current risk as it may already exist. It also centers on preparing staff to understand and deal with a security process. As such, it is the hardest to provide generic guidance that applies equally to

[8] http://csrc.nist.gov/publications/nistir/ir7564/nistir-7564_metrics-research.pdf

every organization. None the less, there are common practices that make a lot of sense in fine-tuning an organization to augment it with security.

Many policy and compliance practices are industry specific, but creating a risk sign-off process can be a hook to expand compliance efforts later. For releases, risks are enumerated and an appropriate risk owner signs off on the suitability of the risks for the given application under the existing business conditions. The first step in adequately mitigating risk is to realize it exists and have an enumeration of policy and compliance risks, whether they come from regulation (external) or internal business requirements or practices. After that, create a security training program for everyone involved in conceptualizing, analyzing, design, implementing, testing, releasing, supporting or managing any of the previous roles. It can be OK to start with the most technical individuals, but the plan should ultimately be holistic and incorporate all of these roles, to educate them on security threats and how they can be mitigated. Training new employees is another key aspect, so that all employees have the same working understanding of security concepts and techniques, regardless of how new they are.

2.2.2. Intelligence

The Intelligence pillar collects together practices that look at the deeper perspective of organizing security into a product under development. Intelligence is not about a specific product being developed, but understanding security and technical risk, and integrating this understanding into the development process. While Governance prepares an organization to deal with a security process, Intelligence establishes how an organization will go about creating secure products and services in the abstract. It details how to prepare and advise teams to go about creating those products.

Attack Models

Attack models are a structured set of understanding used to grow and advance an organization's ability to analyze and contextualize security and threats. Tasks include building attacker personas of types of people who would attack the organization and its resources, creating a security classification scheme or enumerating top N lists of kinds or classes of security defects. As the attack model practice matures, more advanced tasks include creating a science team to research new kinds of attacks, creating technology or platform specific attack patterns and tying identified attack patterns to specific kinds of testing. Attack models are important to grow the technical depth in understanding security risks, but many organizations get started in simple ways by trying to gather industry best practices and disseminating the information inside the organization.

Security Features and Design

Architecting and designing features to be secure, as well as creating security functionality can be challenging. The security features and design practice provides a set of related tasks to incorporate better security architecture into the design process, as well as create organizational patterns. Initial tasks include publishing the set of security features that are available from a security team and are already pre-vetted. Functionality like authorization and authentication which can be difficult to build without specialized security knowledge, lends itself to being built in this fashion. Once functionality can be encapsulated into a component or framework, the security team can test it and then evangelize its use. Other security feature and design tasks include connecting the security team with the architecture team, when an architecture team exists and providing more complete middle-ware or frameworks and libraries that will let developers who may not be security focused to simply fall into the pit of security success. Secure default configurations in these frameworks are an oft-used way to help increase an organization's security stance.

Standards and Requirements

In preparing to build products securely, the final practice of the intelligence pillar is standards and requirements. This can be closely related to Compliance and Policy from the Governance pillar in some organizations, but in others may be discrete individual areas. For the purposes of this technical report, Compliance and Process will deal with external pressures that are exerted on the development process, while Standards and Requirements are guidelines or necessities that come from internal pressures like business decisions or technical objectives that are laid out for a project. None the less, the distinction between the two areas can be nuanced and in many organizations they can be considered the same, especially when regulations are relatively minor or codify a set of industry best practices the organization would follow anyway.

Many organizations start by creating a set of security policy that guides how security sensitive choices are made when developing hardware and software. Common targets include the use of cryptography, when auditing functionality is included or how machines can be connected to different network segments. Practical decision processes may include referring risk questions to a central security group, performing a threat modeling exercise to understand the risk, or requiring executive buy-off when making trade-offs that sacrifice security for some other important business objective. Secure coding standards, with a list of banned APIs or a set of ways in which particular components or sub-components are called or included in a project are also common. As an organization matures, a security review board may be setup to provide specialized knowledge when overseeing a particular kind of development. Again, cryptography is a common topic used here, because cryptography requires such specialized knowledge. The use of third party libraries is another common area where additional scrutiny is appropriate, especially in some regulatory environments like those dealing with life-safety equipment or life-sustaining hardware.

Where to Start

The Intelligence pillar prepares a team for creating, but with a focus on the technology rather than the organization or people. The most important focus for an organization beginning their adoption is to have a clear way of thinking about security and security decisions. Looking at standards and requirements to help guide a decision making process about the minimum level of security that is appropriate is the initial focus. Understanding a maximum level of risk, or the minimum level of security that is appropriate can be one of the most challenging aspects of adopting a security process for new organizations, and is frequently under characterized by management and over-characterized by new security engineers. Formalizing how security decisions will be weighed and what risks will be a focus can help streamline the decision process.

Next, the engineering process starts with developing a set of capabilities to engineer great security features as well as understand the security properties or threats in all features whether their focus is security or not. Much of this practice will grow as the organization’s security capability grows. The most important place to start is to identify a person or set of people who will take responsibility for starting the security thinking. When possible, they should participate in architecture and feature discussions. These people may be embedded in a particular engineering team or may be shared among several engineering teams. These people can be responsible for distributing some industry information on current risks and threats, which is the appropriate initial investment in attack models until an organization matures further.

2.2.3. SSDL Touch-points

It may seem odd, but the third pillar in BSIMM is finally focused on creating technology. This speaks to the complexity of creating secure technology and the need to embed security thinking across the organization rather than making it a single individual’s responsibility. The SSDL Touch-points zeros in on the software development process, and correctly integrating security tasks into the overall process. It was originally a process recommendation from Cigital that was separate from BSIMM but was incorporated as the third pillar. It narrowly focuses on engineering a specific product and what practices can best support creating products securely.

Architecture Analysis

Architecture analysis deeply incorporates security into the design and architecture process, when turning business ideas into an implementable plan. Many organizations that have an existing revenue base with products start by asking each product or development effort to answer a questionnaire to begin to categorize risk and focus on the highest risk areas. Having the security team participate in architecture and design review is also a common initial practice. However, this may be unsustainable and once the architecture or engineering teams gain more experience in doing security analysis of an architecture, the routine engineering practitioners take over the responsibility. Organizing also has an important role here, to provide the capability to track answers to questionnaires and to understand security impact of different designs by having central websites or software to manage the process.

Code Review

Code review is one of the key ways to drive organizational consistency and to identify implementation issues before shipping them. Many code review practices are common among organizations with strong security programs, such as mandating code review of all new code or of security sensitive functionality. Using automated static analysis tools, which operate on the source code, is also frequently used, as is incorporating a top N list of code defects as a cognitive helper to consider for code review participants to review before they begin code reviewing.

Security Testing

The final practice of the software creation pillar is that of security testing. Once a project has been architected and then implemented, it is ready to be tested. Security is routinely incorporated very directly into the quality assurance or verification stage, and it is common for the uninitiated to think only of security testing when considering security tasks that are part of the construction process. Initial security testing frequently focuses on security features like authentication or encryption to verify the functionality is implemented correctly. Boundary conditions and edge testing are then included for the rest of the functionality. But as security testing matures, more direct testing efforts to identify security defects like penetration testing and fuzz testing are brought to bear. Security testing in many cases is the only opportunity to review the product when it is relatively complete to ensure that the pieces fit together in a secure fashion.

Where to Start

Because the third pillar is focused on building products, getting started begins with getting some individuals assigned to a particular product to start performing security tasks. The most successful security engineering efforts usually have dedicated people, though depending on the organization or team it may be unreasonable to dedicate an entire person to such one product and may be shared among several instead. At the very least, a product needs someone who is responsible for security tasks and thinking, even if that is not their only focus.

Once staff is assigned, they can start with performing security tasks associated with architecture, code and security testing. When possible, security engineers should participate deeply in architecture discussions, especially those concerning protocols, security features like authentication, client/server architecture or distributed systems and consider the architecture from an end-to-end perspective. When an organization has adopted an agile methodology and eschews long-term architecture planning, consider a set of security architecture backlog items added to iterations to review the architecture on a recurring basis, even if it is not every sprint. As a side-note, if agile is the preferred methodology and it seems impossible to map the monolithic BSIMM process onto an agile methodology, refer to how Microsoft broke up the SDL to map into the Agile methodology[9]. The Agile SDL process may be more appropriate, but also the same methodology can be used to separate BSIMM or another process into one-time, recurring and every-sprint tasks to incorporate them into Agile.

For code review and security testing, initial efforts are usually very risk directed. It is unlikely that an organization that is beginning to adopt security is going to have a complete understanding of their risk profile from the beginning, however it is more important to begin performing security tasks than it is to have a complete understanding. Engineers should use existing understanding of an application and organization’s risk profile to focus code review and testing on the highest security risk areas and refine the focus as understanding of risk improves.

2.2.4. Deployment

The final pillar emphasizes the importance of holistic thinking when approaching how to effectively mitigate security issues. Many technology producers incorrectly think that security needs to address the creation of technology so that it is made securely. However, it ignores an important part which is how the technology is used and adapted by customers and those responsible for running it. The most hardened technology can be made impotent by an insecure configuration, and some of the weakest products can be used with relative safety by protecting them well in a layered cocoon of additional technology controls.

Penetration Testing

Penetration testing in many organizations straddles the line between creation and operationalizing technology. In many companies, pentesting is used before a product is shipped. However, performing pentesting in an environment that matches what the product deployment will use, or using the production environment can yield additional security defects that can remain hidden when focusing narrowly on the created product instead of the contextual environment it is deployed into. Using pentest tools by other engineering staff and feeding pentest results back into the development process is key. When using penetration testers, whether they are internal or external, they should be provided with all available resources and artifacts included design diagrams and documentation, feature descriptions, source code, available test environments and custom tools to look at the application. The purpose of the penetration test is to enumerate as many ways as possible that the product is insecure, not to mimic the pain and difficulty a legitimate attacker goes through to pull off an exploit.

Software Environment

Monitoring the deployed environment is the second practice involved in managing deployed technology. This includes host and application configuration review and monitoring. Logs can be aggregated across different technology systems to get a more complete picture of deployed and operationalized technology. Using code signing and code protection can further harden systems to prevent unauthorized applications from running. Advanced organizations may use behavior analysis techniques, which consists of capturing a set of data to create a normative model of how an application, system or set of systems operate. Once

[9] http://www.microsoft.com/security/sdl/discover/sdlagile.aspx

the model is well understood, deviations to the model can be tracked and investigated, in many cases leading to ongoing security incidents as well as other kinds of operational situations that should be reviewed.

Configuration and Vulnerability Management

The final practice revolves around managing configuration changes and identified vulnerabilities. The place to start for almost every organization is to put together an incident response plan, with the understanding that it is not a matter of if but when a security incident will occur. More than ever before, organizations face a litany of different threats and threat actors. Once a plan is put in place, additional tasks include preparing an inventory of running programs and applications, creating the necessary feedback loops to incorporate findings from running technology into later revisions and tracking defects identified operationally throughout their fix cycle.

Where to Start

The deployment pillar completes the process by emphasizing the importance of managing security risk even after products have been released or completed. This can be a significant investment in continual monitoring of risk, including patches and updating. One of the first places for a producer of technology products to begin is to create a public presence for their security team and a contact point for an organization. Though not spelled out in the BSIMM model, which focuses a little more in the deployment pillar on how to operationalize technology for a company rather than a vendor, it is still a very important part of creating technology. This will be the first point that responsible members of the community can reach out to an organization to notify them of security issues that may be found in production systems. By having a portal and public presence, the community will have an easier way to notify you. When it is difficult, many people may not share the information as it takes too much of their time.

2.3. Maturity Model Conclusion

There are several different models for how to adopt a security process. Though this report details BSIMM, choosing any of them is better than none. It is important to begin practicing, and the where to starts give hints that organizations large or small that are new to security can begin in creating a process capable of creating hardware and software products while better reducing and mitigating security risk. No process or product is perfect, and nothing is ever completely secure. Hiring or reassigning interested staff to the security effort is an important first step, and they can begin the process of preparing an organization to fully epitomize security thinking while altering how the organization thinks about product initiatives to better incorporate security. They will also become responsible for helping to facilitate the development process and then monitoring products once they go out in the world. Organizations should get started today in protecting their products, their customers and their own reputations which can strongly influence their bottom line. In the increasingly connected world security is more important than ever to safeguard assets, resources and even human lives.

3. Vulnerabilities and Mitigations

3.1. Introduction

In the previous section this report discussed the security process or security maturity model. This section will cover the vulnerabilities and related mitigations in an example solution, using a hypothetical product called the *MeshPet 2000*.

> How the MeshPet hypothetical attack scenarios play out will be discussed in highlighted boxes such as this.

The System Under Review

The marketing team describes the MeshPet in a press release as follows:

> "The *MeshPet 2000* is the next generation of pet monitoring solutions. Leveraging IoT concepts the *MeshPet 2000* allows users to monitor location and health data for their menagerie of animals. Stylish pet collars provide a bevy of sensors including GPS, accelerometer, light, temperature, heart rate and a camera as well as a mesh networking module. The innovative mesh networking protocol allows pets to roam far from the *MeshPet 2000* basestation while still reporting data back via their fellow pets. The *MeshPet 2000* provides both web and mobile applications for administration and analytics. Never wonder about the health and safety of your pets with the new *MeshPet 2000*."

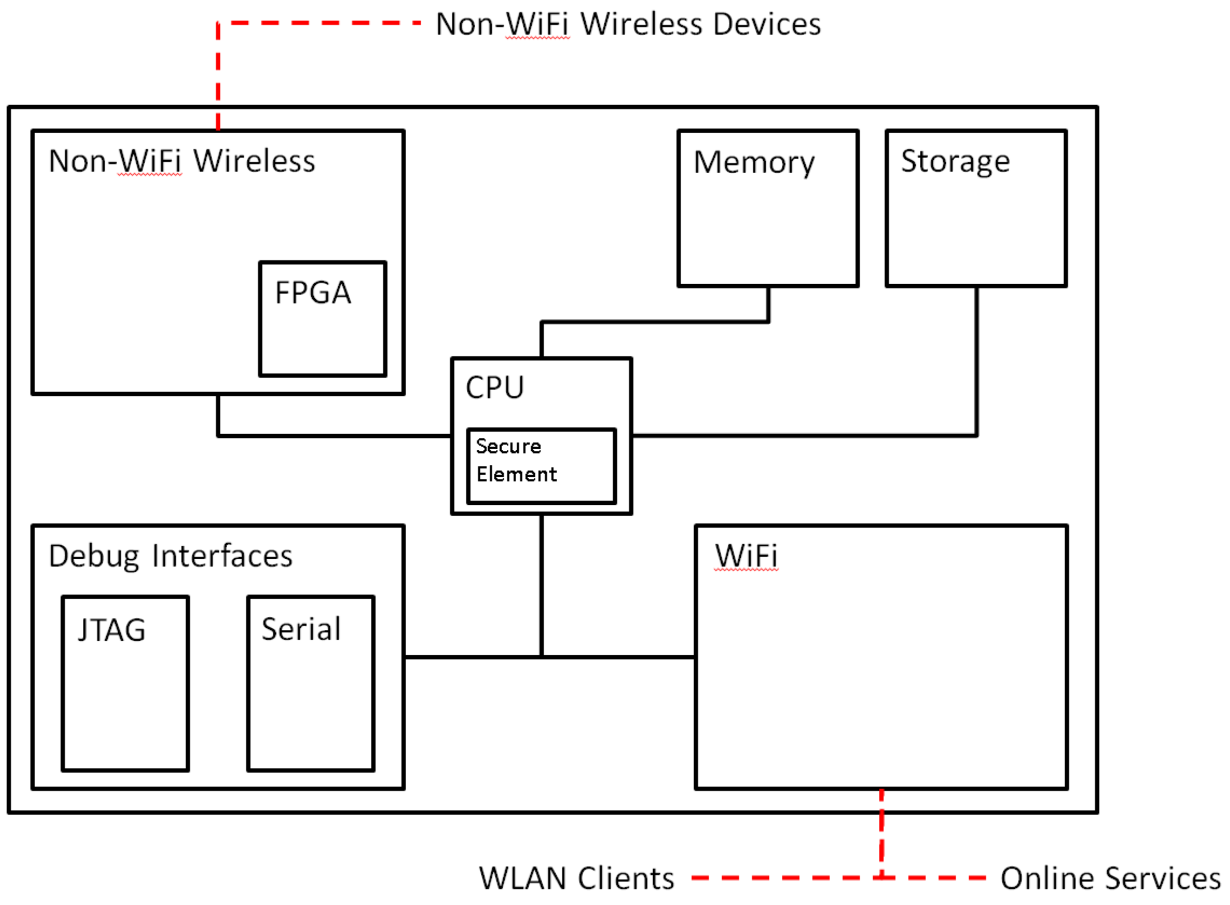
Security of this system, however, was compromised in a number of ways after it was released. The rest of this section will discuss the threats the system faced, how it was compromised, and how the development team could mitigate each threat.

Architecture

The *MeshPet 2000* is a combination of multiple embedded devices, multiple mobile applications, multiple web applications and web services. For the purposes of this discussion the focus will be on the *MeshPet 2000* basestation and its connections to external devices (including the *MeshPet 2000* pet tracking units) and services.

The *MeshPet 2000* basestation is a standard embedded device running a commodity operating system. It provides multiple external interfaces including WiFi and a proprietary mesh networking protocol that leverages trust-on-first-use principles to build a secure mesh network. Internally the device has a full featured Central Processing Unit (CPU), including a secure element. This secure element may be a dedicated secure coprocessor, such as a Trusted Platform Module (TPM), or secure extensions provided by a system on a chip (SoC) such as ARM TrustZone.

The *MeshPet 2000* basestation contains standard random access memory (RAM) and persistent storage as well as multiple debug interfaces such as serial and Joint Test Action Group (JTAG), also known as IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. The main application CPU communicates with two chips to provide the two wireless paths. For upstream communications to the Internet/cloud/apps, there is a WiFi SoC (System on a Chip) on the system bus. For downstream communications to the *MeshPet* pet collars, a second SoC provides a proprietary non-WiFi wireless access over an internal wired Ethernet network. There will be some minimal discussion of the communications between pet collars inter-device *MeshPet 2000* pet tracking unit communication but the underlying implementation of the pet tracking units is not specified in this report.



3.2. Network Vulnerabilities and Mitigations

Sniffing

Network sniffing, or eavesdropping, occurs when an attacker monitors network traffic in order to discover sensitive data such as login credentials, configuration data or personally identifiable information. There is a wide variety of commercial and open source tools available for network sniffing[10]. The use of encryption does not always safeguard data in transit. Encryption schemes that use weak algorithms, or that do not properly authenticate parties allow attackers to compromise communication.

Network sniffing requires an attacker in the path of the communication. This is trivial in the case of a shared Ethernet or WiFi network, where the attacker can gain direct access to the network traffic. Attacks over the public Internet are non-trivial, but evidence indicates that both nation states[11] and private individuals[12] are capable of remotely sniffing Internet communications without direct access to the user's local network.

In the context of the *MeshPet 2000* network sniffing could cause significant issues. Use of the *MeshPet* 2000 basestation on shared or unprotected WiFi networks could lead to the real-time disclosure of confidential data about owners, and the actual pet locations. This location data could be leveraged by an adversary to kidnap valuable pets leading to a significant financial costs to the *MeshPet 2000* customer and potential legal costs for the *MeshPet 2000* manufacturer.

SSL or TLS?

SSL was original developed by Netscape and released in 1995. While either protocol is better than having no application layer security at all, all versions of the SSL protocol have known security vulnerabilities. Use of the latest stable version of the TLS protocol is strongly encouraged.

There are multiple countermeasures that are useful in preventing network sniffing. If possible strong physical security, including network segmentation, can alleviate some issues. However, it is difficult to construct and maintain a strong perimeter. When possible, the use of strong encryption and authentication will prevent network sniffing. Transport Layer Security (TLS) and Internet Protocol Security (IPSec) are protocols that can provide confidentiality, integrity and authentication. At the time of this report all versions of the Secure Socket Layer (SSL) protocol suffer from deficiencies and should be deprecated in place of TLS.

[10] https://en.wikipedia.org/wiki/Packet_analyzer#Notable_packet_analyzers
[11] http://archive.wired.com/science/discoveries/news/2006/04/70619
[12] https://www.defcon.org/images/defcon-16/dc16-presentations/defcon-16-pilosov-kapela.pdf

3.3. Host Vulnerabilities and Mitigations

Authentication Failures

One of the most common ways embedded devices are compromised is through the use of default passwords for configuration or remote management interfaces[13].

In the worst instance default account names and passwords are in use, sometimes for services that are made available remotely. In the case of embedded devices these credentials may be hardcoded into the device. Any attacker able to reverse engineer the device may be able to extract credentials that can be used to compromise every other device. These hardcoded credentials may also leak via technical manuals that are published online.

The *MeshPet* basestation includes a web administration portal. This web administration portal does not perform proper authentication or authorization. Any user with access to the web interface may act as an administrator. This would allow an attacker to subvert the privacy of *MeshPet* end users and could have a significant impact on the reputation of the *MeshPet* service.

Online brute-forcing occurs when an attacker is allowed hundreds, thousands or more attempts at logging in. Attackers will leverage published lists of common usernames and passwords, or will develop targeted password lists for specific individuals or manufacturers. Typical guidance has been to enforce “strong” password quality to make brute force attacks more difficult. However, this advice is of dubious utility[14].

User: Admin

Password: password

Passwords should be required to be changed upon set-up of new devices

A recessed reset button or a special button sequence can restore the default password, if needed for support purposes. This is a trade-off between security and ease-of-use, but unfortunately, the easiest solutions are often the least secure.

System implementers can best prevent online brute forcing attempts with rate-limiting techniques. The most common techniques are exponential back-offs, which increase a timeout after each successive login failure, and mechanisms that require human interaction, such as a Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA).

[13] http://arstechnica.com/security/2012/10/dsl-modem-hack-infects-millions-with-malware/
[14] http://research.microsoft.com/pubs/227130/WhatsaSysadminToDo.pdf

Attack Surface Minimization

The attack surface of the *MeshPet 2000* basestation is the collection of the different interface points, or attack vectors, where an attacker may access the device. It is not uncommon for embedded devices to go to production with unnecessary services enabled.

One example is Telnet. This remote access service may be part of the default operating system that was used as the basis for developing the product. It can be useful in debugging the product during development. But if it is still enabled when the product is shipped, Telnet is a serious vulnerability.

Any service that accepts user input of any kind can be an avenue to compromise. Disabling these services is the best line of defense. If the services cannot be disabled, firewall configurations and IPSec filters can be used for defense in depth.

> The *MeshPet* basestation makes available a range of services on unique ports. The many available services are difficult to secure due to the diversity in the code base and difficulty staying in sync with upstream open source projects. A network attacker leverages this opportunity to use a known exploit on one of the network services to execute arbitrary code on the device. The attacker is now able to export any unprotected device secrets or subvert the privacy of the authorized user.

Why is Telnet a vulnerability?

Telnet protocol is a protocol that allows bidirectional interactive access to computer systems over the Internet or local networks. The Telnet protocol does not provide confidentiality or integrity of communications. In the presence of an attacker, use of the Telnet protocol can lead to complete system compromise. There are far superior alternatives available, such as SSH.

Regardless of the necessity of any particular network service, locking down network interfaces with an appropriate firewall configuration is an essential component of system hardening. Measure the attack surface[15] made available by a device and take whatever steps possible to reduce that attack surface.

System Identification

Attackers that stumble upon devices, without physical access, must identify (“fingerprint”) the device before beginning a targeted exploitation of the device. Attackers may use techniques like banner grabbing to study a target. Servers and devices should be configured to present a minimum of such information, to restrict access to services to authorized network traffic, and to eliminate unnecessary services.

> The *MeshPet* basestation makes available a range of services on unique ports. This combination of interesting services on unique ports makes the *MeshPet* basestation easy to “fingerprint” on a network. That is, the attackers are able to identify the system as a *MeshPet* as opposed to other devices on the Internet. Knowing that the target system is a MeshPet, attackers can search the Internet for known vulnerabilities in the *MeshPet* solution or its constituent parts.

Obfuscating the system identity by minimizing available information is a valuable defense in depth tactic, but it should not be viewed as a security boundary per se. Systems leak an incredible amount of information, including subtle parameters in basic network communications. This information can be used to fingerprint a system. However all reasonable efforts should be taken to hide the system identity. This ensures that unauthorized users cannot quickly deploy off the shelf exploits to compromise a system.

[15] https://www.cs.cmu.edu/~wing/publications/Howard-Wing03.pdf

Common steps include ensuring that web server banners do not report version numbers and reducing the number of services available on the system.

Unauthorized access

One path by which a platform can be breached is via weak internal communication mechanisms. This issue was the undoing of the content protection scheme in the original Microsoft Xbox. The original Xbox used a fast, narrow, parallel data bus called Hypertransport for internal communication. This mechanism was cutting edge in its day and was considered unfeasible to monitor. What the designers did not account for was the curiosity of an undergraduate at the Massachusetts Institute of Technology, named Andrew "Bunnie" Huang[16], with access to high speed hardware bus monitoring equipment. In a similar fashion many equipment manufacturers will rely on simple plain text serial transports, Ethernet transports or similar without taking steps to safeguard sensitive data.

> The *MeshPet 2000* uses a device specific shared secret in order to gain access to *MeshPet* cloud APIs. For cost and capacity reasons it is important that these APIs only be accessed by authorized devices. Unfortunately this API key is transmitted internally over a clear text bus. Enterprising users with common test equipment extract their device specific API keys and sharing them in Internet forums. In order to combat this misuse the *MeshPet* company is required to invest significant development cycles in a fraud monitoring solution.

Developers have two options for protecting internal communications. The developers can use a secure internal transport mechanism or they can ensure that sensitive data is not transmitted over internal transports in clear text. In this specific instance the *MeshPet* developers may have considered using a challenge and response protocol for API authentication where the challenge was processed in a secure element.

All applications require trust in the underlying platform on which the application is built. One way to preserve this trust is through the use of secure boot technologies. By creating a secure boot chain, and protecting executable code from the boot loader all the way up to the running system, trust can be propagated from a very small code base to a large and feature rich code base. The primary reason for securing the boot process is to prevent the propagation and persistence of malware and backdoors.

> In an unfortunate series of events the *MeshPet 2000* shipped with a combination of vulnerabilities that allowed an attacker to execute arbitrary code on the basestation. The attacker leveraged this initial access to send a compromised firmware update to the device. In doing so the attacker was able to ensure that their access to the device would be preserved after the device has been restarted or power cycled. After this compromise the only way for the end user to regain control of the device is through some out of band mechanism for updating the device firmware.

A secure boot chain requires a mechanism for propagating trust from the bottom all the way up to the running system. The system must start from a "known good" point and authenticate subsequent step in the chain. One example of a secure boot mechanism, that preserves the ability to update the system, can be found in Appendix A: A Secure Boot Mechanism.

[16] http://bunniefoo.com/nostarch/HackingTheXbox_Free.pdf

It is not uncommon to find that debug modes are left active in production releases of devices. When this occurs end users can gain privileged access to connected device platforms. These debug interfaces include local interfaces, typically serial and JTAG interfaces; and remote interfaces.

Serial interfaces may give access to an interactive shell as a normal privileged or administrative user. Sometimes this privileged access is weakly protected by monitoring for specific conditions during boot. For example, boot code may monitor device buttons for a specific sequence that enables the serial console. This style of obfuscation is not a proper security boundary, and may be discovered or even disclosed by company staff to "trusted" partners.

Unauthorized JTAG access can be extremely powerful giving access to the contents of memory and CPU state. Although JTAG may seem like a tool reserved for the hardware designer in a corporate lab, many universities and individuals have JTAG explore/exploit capability. Remote access allows attackers to reach many devices without needing any physical access.

The *MeshPet 2000* basestation goes to production with an error in configuration management. A remote debug interface with a static password was left enabled, as was a serial console interface. Exploiting the serial access to the *MeshPet* device, an attacker discovers the remote debugging interface and the associated hardcoded password. The attacker then scans the Internet looking for *MeshPet* devices and sets up a web site allowing anyone to locate and exploit *MeshPet* basestations. This website is reported on by major media outlets leading to significant damages to *MeshPet*'s finances and reputation.

Automated Exploiting of Debug Interfaces

JTAG interfaces can be very time and labor intensive to explore and exploit but automated mechanisms for accessing these interfaces have become available. One inexpensive commercial tool is the Jtagulator[17] which can automatically probe exposed electrical contacts and establish interactive access with both serial and JTAG interfaces.

Unless absolutely required, ensure that no debug modes are left active in production releases. If debug modes are required in production releases protect them using effective authentication and authorization mechanisms. Several such mechanisms are provided for JTAG implementations by major component manufacturers.

3.4. Application Vulnerabilities and Mitigations

Ineffective Input Validation or Output Encoding

Applications accept and interpret a wide range of user input and the *MeshPet 2000* basestation is no different. The basestation supports no fewer than two wireless networking protocols, has multiple web interfaces, a web service interface and several debug interfaces. Each input is subject to its own set of security risks. The interpretation of user data as code is one of the central problems in computer security. Some common vulnerabilities associated with user input include memory corruption, Cross-Site Scripting (XSS), Structured Query Language (SQL) injection, path canonicalization and character canonicalization. Each of these is considered in the following discussions.

There are many forms of memory corruption that affect modern systems. Stack based buffer overflows[18] are a common type of memory corruption and primarily affect systems programmed using native code

[17] http://www.grandideastudio.com/portfolio/jtagulator/
[18] https://en.wikipedia.org/wiki/Stack_buffer_overflow

languages such as C and C++. A buffer overflow is a software flaw wherein an area reserved for data is smaller than the data copied to that location. When exploited, the user-provided data will overwrite program code. Allowing the attacker a way to rewrite product code may lead to a denial of service or remote code execution. There are multiple other forms of memory corruption, such as heap overflows and the "double free".

> The *MeshPet 2000* shipped with a vulnerability in the mesh networking software. This memory corruption vulnerability would allow anyone within range to send a malformed mesh network packet that would exploit a stack based buffer overflow. Once the attacker exploits this memory corruption vulnerability they are in a position to man-in-the-middle communication between the basestation and the pet monitoring collars. An attacker may leverage this position to notify a *MeshPet* user that their pets have are deceased in a remote location. This "prank" could cause significant emotional distress for the *MeshPet* user and lead to significant damage to the *MeshPet* reputation.

Prevention of memory corruption vulnerabilities starts with the selection of the language in which applications are developed. Choosing a managed language (such as Java or C#) or an interpreted language (such as Python or Ruby) can mitigate most memory corruption vulnerabilities. The underlying runtime or interpreter may still be vulnerable to memory corruption but those issues can be more difficult to exploit and are more easily remedied through patch management.

If code must be written in a native language the next strategy is to prohibit dangerous functions. There are multiple lists of banned function calls, including the Microsoft SDL list[19]. Finally system implementers should take advantage of any platform functionality that may reduce exposure to memory corruption vulnerabilities. This functionality may include features such as stack cookies or canaries; non-executable stack and heap protection; Address Space Layout Randomization (ASLR); safe structured exception handling; function pointer obfuscation and more.

In the context of web applications the vulnerability most similar to memory corruption is Cross-Site Scripting (XSS)[20]. When user input is included in server output, in an unfiltered and un-encoded fashion, it is possible for a malicious user to include executable code in the form of JavaScript, CSS or other rich content. Allowing arbitrary users to modify the client side behavior of a web application violates the same-origin-policy[21]. These vulnerabilities can lead to a loss of web application session identifiers, user credentials and any data associated with the application. These vulnerabilities also allow attackers to take any action for which the user is authorized.

> The *MeshPet 2000* basestation includes a web management console. This management console does not perform any input validation or output encoding of URL parameters. An attacker is able to formulate a specific URL that when clicked by a *MeshPet* user will cause JavaScript to execute in the users' browser. This specially formulated URL is distributed via email, instant messages and forum postings. Any user that mistakenly clicks on the malicious link confers control of the *MeshPet* basestation to the attacker.

Preventing XSS is best performed in two ways. Both input validation and output encoding are required for a robust XSS prevention. The strongest form of input validation is whitelist filtering, in which the developer specifies exactly what input is allowed. An example of this might be numbers 0 through 9 and both

[19] http://msdn.microsoft.com/en-us/library/bb288454.aspx
[20] https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29
[21] http://tools.ietf.org/html/rfc6454, section 3.5 (Conclusion)

uppercase and lower case letters. Whitelist filtering is superior to specifying bad input, so called blacklist filtering, as the set of dangerous characters is large, difficult to express and constantly evolving.

In addition to filtering input, encoding output is a very effective strategy for avoiding XSS. It is essential that user input is encoded into a form appropriate for the context it will be presented. For example, if the output includes special characters like ‘&’ or ‘>’, these can be translated to “&amp;” and “&gt;” (“character canonicalization”) to keep them from being interpreted as special characters in an HTML context[22]. There are many contexts and encoding strategies including HTML, URI, CSS, XML and more. Robust input validation and output encoding is best accomplished using a library provided by a trusted third party such as Microsoft[23] or OWASP[24].

Another case of data interpreted as code in web applications is SQL injection[25]. This can occur when user input data is used by the server application to build a SQL query. When user input is placed directly into a SQL query it is possible for the user to modify the functionality of the SQL query. This may lead to an escalation of privilege or even arbitrary code execution on the system hosting the database[26].

> The *MeshPet 2000* basestation includes the above mentioned web management console. The attacker again builds a specific URL that exploits the lack of input validation or output encoding on the website. However, this time, the attacker does not use JavaScript; instead he encodes a sequence with carefully-selected special characters. When clicked by a *MeshPet* user, the specific URL will cause SQL to execute on the user’s basestation. The attacker is then able to take control of the *MeshPet* basestation and subvert the users’ privacy. The attacker may even leverage this position to compromise other devices on the users’ network.

There are a wide variety of mechanisms available in order to prevent SQL injection. These range significantly from the use of prepared statements[27] (also known as parameterized queries), through the use of stored procedures, to the “escaping” of all user supplied input and finally to the use of object relational mapping systems or other non-SQL mechanisms.

Inappropriate path canonicalization[28] can lead to a variety of issues including the ability of an attacker to upload a file to an unauthorized location or to access a file without authorization. One form of these attacks goes by the name directory traversal (or path traversal) attack. Path canonicalization is the more general form of mitigation for directory traversal attacks, as it includes not only the case where operators are included in a path but also the case where unforeseen character encodings are used to confuse application servers.

> The *MeshPet* basestation allows end users to upload photographs, or avatars of their pets. This mechanism performs faulty path canonicalization. The *MeshPet* basestation does not check or convert the strings in the names of uploaded files; it uses the user-supplied filename directly. Malicious users begin crafting filenames that include strings like “`../../init`” to try to reach system directories above the intended user directory.

[22] https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet
[23] http://msdn.microsoft.com/en-us/security/aa973814.aspx
[24] https://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project
[25] https://www.owasp.org/index.php/SQL_Injection
[26] https://xkcd.com/327/
[27] https://en.wikipedia.org/wiki/Prepared_statement
[28] https://www.owasp.org/index.php/Canonicalization,_locale_and_Unicode

In this manner they are able to insert executable code in a system directory where the basestation's embedded web server will launch it. This gives them arbitrary access to the device. This level of access allows any device level secrets, not adequately protected, to be exported. This level of access also allows users to circumvent any application level restrictions *MeshPet* may have in place.

Effective path canonicalization can be accomplished in a variety of ways. The most effective strategies enforce a "document root", or a folder from which content must be served. Upon request for a file or directory the application should build a full path to the resource and normalize all characters. Validate or enforce the document root at this point. These techniques can be platform provided or implemented by the developer.

Weak Configuration Management

Administration interfaces are frequently available on either the Local Area Network (LAN) or Wide Area Network (WAN) interfaces. These administration interfaces take a variety of forms, often that of a web application. All too often these interfaces are inadequately protected and can be accessed using default credentials, forced browsing or other authentication and authorization bypasses. These bypasses may allow an attacker to subvert end user privacy and change sensitive settings on the device. Device configuration data frequently contains sensitive data including passwords or valuable metadata about users.

A casual acquaintance of a *MeshPet* user gains access to the user's WiFi network. Leveraging this network access they exploit an authentication weakness in the *MeshPet 2000* basestation—the admin account and password are the same on all *MeshPet* products—and gain access to the administrative console. Using this level of access the attacker adds an email address to the list of email addresses to be notified when a pet leaves or enters the home. Knowing that the owner always brings the dog into the house before leaving the attacker uses this knowledge to break into the home while the owner is away.

Ensure authentication and authorization mechanisms are robust. Ensure systems have mechanisms in place to prevent online brute-force or dictionary password attacks. Use effective web application session management techniques and test the application to provide confidence all sensitive interfaces are protected. Effective web application session management will protect against HTTP cookie guessing attacks, HTTP cookie discovery attacks and HTTP cookie setting attacks. An example of effective web application session management is the use of a large, un-guessable client side HTTP cookie value protected with available platform features, such as the HTTPOnly and Secure flags, as well limited with a properly scoped domain and expiration. This value would provide an index into a state table on the server side and the value would be refreshed on all authentication operations. More details about secure session management can be found in Chris Palmers' paper *Secure Session Management With Cookies for Web Applications*[29].

In the rush to get products and services to market it is not uncommon for developers to use over-privileged process and service accounts. In the worst example of this vulnerability class a developer may run all applications under a system's administrative, or root, user account. This ensures that system access restrictions do not cause functional bugs, which seems like a good thing. However, the impact is

[29] https://www.nccgroup.trust/us/our-research/secure-session-management-with-cookies-for-web-applications/

that *any exploited software will be running as an administrative user.* Running individual software components with accounts that have constrained privileges is an important defense in depth mechanism.

> The *MeshPet* basestation shipped, in its recommended configuration, with an Internet accessible web server running under the system's root account. This web server was vulnerable to a memory corruption vulnerability, described early in this document. An attacker exploiting the memory corruption vulnerability obtains root level access to the device without any further steps required. This root level access is used to install additional exploitation tools required for exploiting other devices on the *MeshPet* user's network. This exploit is automated and 80% of *MeshPet* devices in production are compromised. Each device requires in person physical intervention in order to evict the attacker and restore end user control.

Create process and service accounts specific to their purpose and with the only the privileges required. For example, run a web server under a dedicated web user account while running a batch service under a specific batch service user account.

Sensitive Data Exposure

Sensitive data exposure can take a variety forms. Any time valuable user data is stored or transmitted in an insecure fashion there is the potential for sensitive data exposure. Persisting (storing) data in a clear text backup may inappropriately expose user data. Transmitting user data using clear text protocols such as HTTP, SMTP, Telnet or FTP may expose data. Using weak cryptographic protocols or relying on obfuscation in lieu of encryption can lead to data loss. Developing web applications that encode sensitive data in HTTP headers may lead to a breach of end user privacy. The possibilities for sensitive data exposure are many and these are but a few examples.

> The *MeshPet* mobile application is designed in such a way that it attempts to poll the *MeshPet* cloud service for pet information at a set time interval. These requests are made in plain text HTTP and they are performed whenever the application has Internet connectivity. The requests and responses include names of pets, their current geographic location and other personal details. Any network eavesdropper could obtain this information. Even more troublesome for *MeshPet* the vulnerability is discovered by a major media writer researching connected device security and privacy issues. The issue is highlighted in online, print and broadcast media causing severe damage to the *MeshPet* brand.

Begin protecting sensitive data by inventorying the sensitive data the application handles and considering the threats to that data. If any specific piece of sensitive data is not necessary stop collecting, persisting and transmitting that data. Information the application does not handle cannot be stolen. When required to collect, store or transmit sensitive data ensure secure techniques are employed. This may include using strong encryption for data storage and transport protection, such as TLS, for data transmission.

Cryptographic Weaknesses

The process of creating keys for cryptographic operations is frequently referred to as key generation. It is essential that key generation is performed in a sound and secure manner. Anything less can lead to weaknesses that vary from theoretical to complete failure. There are innumerable ways to insecurely generate cryptographic keys. Of specific interest to the readers of this report is the work of Arjen K.

Lenstra et al in the paper *Ron was wrong, Whit is right*[30]. These researchers were joined by others, including Nadia Heninger[31], in studying a substantial data set of X.509 certificates, used for SSL/TLS. Summarizing the different groups' results, they found:

- That 0.02% of these certificates were generated poorly and offered absolutely no security
- That 0.5% of certificates gathered were exploitable
- That as many as 1.7% of collected X.509 certificates were generated using poor sources of entropy and may be exploitable.

Specifically called out in the Heninger paper are embedded devices, which have limited access to good sources of cryptographic entropy.

The research in the Heninger paper pales in comparison to the incident in which the Debian Linux distribution's version of OpenSSL produced keys with a drastically reduced amount of entropy[32]. On any specific computer architecture the OpenSSL was limited to 32767 possible random number streams. This error made brute force attacks on Debian generated keys trivial, completely undermining the security guarantees of TLS and SSH protocols on this platform.

> The *MeshPet* basestation allows remote access via the SSH protocol for remote administration. In order to protect this network service key based authentication is used. The keys required for authentication are generated when a device is first initialized and are made available to end users via the web interface. Unfortunately the *MeshPet* basestation does not have an adequate source of entropy to use for key generation. This weakness limits the number of values the access key may have. Due to the limited key space employed by the *MeshPet* basestation remote attackers are able to bruteforce access keys and remotely compromise *MeshPet* basestations.

Safeguarding end user devices from compromise via weak key generation is no simple task. May of the devices considered here, including the *MeshPet* basestation, have incredibly limited sources of entropy to use in key generation. They lack the common sources of entropy used on desktop computer systems such as keyboards, mice and spinning magnetic media. Hardware pseudo-random number generators (PRNG) can be procured and embedded within a device. These hardware PRNGs are even included in some system on a chip (SOC) solutions. However the performance of these devices should be carefully evaluated before relying on these devices.

Another, unfortunately common, class of vulnerability is the use of weak or unproven custom encryption techniques. This may vary from the use of obsolete mechanisms, such as the MD5 hashing function or Data Encryption Standard in so called single DES mode, to the use of custom techniques that have no basis in cryptographic research.

> The *MeshPet* service distributes small applications, called applets, from a central *MeshPet* application store. These applets execute directly on the *MeshPet* basestation and have access to all user data. The *MeshPet* application store uses plain text HTTP to distribute the application but the *MeshPet* store developers recognized the importance of protecting the integrity of the applets. To this end they developed a custom cryptographic mechanism based on CRC32 algorithm. Unfortunately for *MeshPet* users the CRC32

[30] http://eprint.iacr.org/2012/064.pdf
[31] https://factorable.net/weakkeys12.extended.pdf
[32] https://wiki.debian.org/SSLkeys#End_User_Summary

> algorithm, even with the custom *MeshPet* modifications, is completely unsuitable for this performance. CRC32 is intended as an error correction code and does not prevent tampering. Due to this weakness network attackers could substitute malicious code in place of a *MeshPet* applet and take control of *MeshPet* end user devices.

The solution to this issue is twofold. First systems should be engineered with the property of cryptographic agility. Instead of hardcoding a specific cryptographic method in code use a development style that allows cryptographic code to be easily updated. Microsoft provides and extensive discussion of this concept on MSDN[33], and those principles are applicable to a wide variety of platforms and tools. The second part of the solution is to choose cryptographic technologies that are secure for the specific purpose. The only way to achieve this is to review available literature on cryptographic technologies. One excellent resource for selecting cryptographic technologies is the NIST Cryptographic Toolkit[34]. This US Government run site lists packages data from NIST standards, recommendations, and guidance to provide end users with the knowledge required to select appropriate cryptographic technologies.

Update System Failures

As described in Section 1.4 all non-trivial systems will have flaws and some of these flaws will have security impact. As these flaws are uncovered they must be remediated. Typically this requires distributing new executable code to update the previously shipped code.

Outside of considerations for ensuring the authenticity of update distributions, the mechanisms used to distribute updated code will drastically impact the update application rate[35] and the security of a platform. Update mechanisms can take the form of an update applied by a service technician, a binary file supplied out of band an applied by an end user, a user initiated update triggered by the end user, an automatic update system that requires user consent and an automatic update system that silently updates and only notifies the user once the update is complete. This is a spectrum of update procedures that can be applied in various scenarios.

> The *MeshPet* basestation shipped with an update mechanism that requires end users to visit the *MeshPet* website, download the latest firmware image and manually apply this image to the *MeshPet* basestation. A critical, remotely exploitable vulnerability is discovered in the *MeshPet* basestation. Because of the amount of end user interaction required to distribute and apply updates the *MeshPet* ecosystem has exceptionally low update application rate. This low rate of update adoption allows attackers to compromise a large percentage of *MeshPet* devices in use leading to severe damages to the *MeshPet* reputation and potential government intervention.

Networked devices with known vulnerabilities require software updates. Failure to provide these updates can lead to the compromise of end user privacy, damage to the manufacturer's reputation, legal action, and government intervention[36].

---

[33] https://msdn.microsoft.com/en-us/magazine/ee321570.aspx

[34] http://csrc.nist.gov/groups/ST/toolkit/index.html

[35] The *update application rate* is the rate at which the updates are actually applied to customer systems in the field, as compared to the rate at which updates are issued. Ideally this should be 100% (of available updates being applied to customer systems).

[36] https://www.ftc.gov/news-events/press-releases/2013/02/htc-america-settles-ftc-charges-it-failed-secure-millions-mobile

In the effort to compel users to update software there have been two very compelling techniques in use that deserve specific attention. Apple drives adoption of OSX and iOS software updates through free updates, the promise of additional features and a reputation for delivering functional updates. While Apple outstrips its competitors in the adoption of these major updates they still have the challenge of users running unsupported versions of their operating systems[37]. Some amount of this unsupported version use is due to obsolete hardware but some amount is due to a failure of an end user to update their current generation hardware.

Google has taken a very different approach with the Chrome web browser. Chrome auto updates its software automatically, and without user consent for each individual update. The only user action required is terminating and restarting the browser, which Google signals to users in their user interface. Similar to Chrome, the Nest thermostat automatically applies updates. Users receive an email informing them an update will be applied in a certain timeframe. No user action is required. The update mechanism implemented will be dependent upon a large number of factors. Regardless of these factors there must be a reliable and efficient mechanism for distributing software updates to end user devices to mitigate known vulnerabilities.

[37] http://www.computerworld.com/article/2853636/os-x-yosemite-sets-mac-adoption-record.html

4. Summary

Incorporating a set of security practices into the development process improves the resiliency of the final product and will be essential to success in the world of the Internet of Things, including those devices consumers take home. Because attackers have become ingenious at finding holes anywhere in the product, a product life cycle process approach—a security maturity model—is strongly recommended.

BSIMM is used as an example in this technical report, but there are other useful security maturity models. Such a maturity model is both a template for change and a set of metrics to measure that change. That is, the maturity model helps answer the question, “What should we do to improve?” and the question, “How well are we doing?”

No matter which methodology is used, one should be used to create a process capable of exerting a downward trend on security issues identified in released technology products.

Attackers have many tools from which to choose. To best protect against these threats, consider the critical threats and mitigations, formatted in the following table as a “Top Ten” list of things product developers can do to enhance connected device security.

Top Ten Developer Actions for Connected Device Security

Action	What's frequently vulnerable	What's exposed	What To Do
1. Prevent wireless sniffing	Data in transit	Personally Identifiable Information (PII) Other sensitive information	Provide physical security, including network segmentation Ensure confidentiality, integrity and authenticity with strong encryption and authentication (e.g. TLS+IPSec)
2. Restrict password guessing	Internal interfaces and resources commonly protected by user ID and password	Solution functionality, data and settings.	Require change of default passwords during set-up of new devices Prevent "brute force" password-guessing with exponential back-off "rate limiting", which increase a timeout after each successive login failure, and mechanisms that require human interaction.
3. Disable unused remote services	Any device service that accepts external input	Pathways to device access	Disable or eliminate all device external input services.
4. Hide system identity	Specifics about the type of device, model, default configuration information, base operating system or interface.	Valuable data that may lead to device or service compromise.	Obfuscate the system identity by minimizing available information as much as possible. Make all reasonable efforts to hide the system identity from an attacker, such as ensuring that the embedded web server does not report version numbers, and reducing the number of services available on the system.
5. Block unauthorized developer-level access	Weak internal communications paths inside the device Boot system Debug modes and interfaces	Internal device function, firmware image, stored and active data.	Ensure that chip-to-chip communications are not conducted in the clear for sensitive information Use a secure boot process Ensure there are no debug modes or interfaces on the shipping product

Action	What's frequently vulnerable	What's exposed	What To Do
6. Validate user input and encode system output	Interpretation of (attacker) user data as code: • memory corruption • Cross-Site Scripting (XSS) • Structured Query Language (SQL) injection • inappropriate path and character canonicalization	Allows an attacker to execute arbitrary code and can lead to a complete violation of any and all system security guarantees.	Enforce the separation of user data and executable code. Examples of this include the use of a managed language (Java, C#, etc.) to prevent memory corruption of code and data.

Action	**What's frequently vulnerable**	**What's exposed**	**What To Do**
7. Protect remote administrative interfaces	Device administration interfaces, usually accessed by LAN/WANHighly-privileged execution modes	The overall security of the device or service. Compromise of the management interface likely leads to a complete compromise of end user data.	Use robust authentication and authorization mechanisms. Prevent online brute-force or dictionary password attacks. Use effective web app session management techniques. Test the application to ensure all sensitive interfaces are protected. Create process and service accounts specific to their purpose and with the only the privileges required.
8. Limit potential for data exposure	Sensitive data stored on the device or transmitted to/from the device	Data stored or transmitted: ..."in the clear" ...obfuscated but unencrypted ...with weak encryption	Identify all sensitive data the application handles and consider the threats to that data. Do not collect/store/transmit any item of sensitive data unless necessary to a documented feature. When sensitive data is required, use secure techniques such as strong encryption for data storage; and transport protection such as TLS for data transmission.
9. Use strong, proven and updatable encryption	Cryptographic mechanisms that use weak primitives or ineffective mechanisms for key distribution.	End user data and the ability to authenticate or issue authorized commands.	Have a qualified expert examine any proposed cryptosystem and ensure it conforms to the best practices currently employed. Provide cryptographic “agility”, the ability to change the solution’s cryptosystem in the face of increasingly advanced attack techniques.
10. Plan for system updates in the field	Devices and systems	Devices & applications that cannot be patched when an exploit is discovered and is being actively exploited.	Design for updates Consider how updates can be distributed with the least impact and involvement of the end-user (e.g., auto-push updates).

Glossary

API	In software development an Application Programming Interface exposes various functions, including inputs and outputs, of an application. There are multiple kinds of APIs including APIs provided by programming languages and web APIs.
ASLR	Address space layout randomization (ASLR) protects software applications from buffer overflow attacks. ASLR implementations program data in a random fashion that increasing the difficulty of exploitation.
Authentication	Confirming the veracity of some attribute of data. Most commonly applied to confirming the identity of a principal in a computing system.
Authorization	The act of specifying or enforcing limits to access within a system.
Availability	A guarantee that the system is in a functional and accessible state.
Banner Grabbing	A method of categorizing network systems or of studying a target system by capturing and looking at the information returned by various well-formed or ill-formed requests.
Bruteforce	An attack technique that attempts to discover the value of a password or cryptographic key by exhausting all possible values.
Canonicalization	Converting data with multiple representations into a single form. In non-security contexts sometimes referred to as standardization or normalization.
CAPTCHA	CAPTCHA is an acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart." Used in order to prevent automated brute-force attacks against computer systems and can be used to provide a rate limiting function.
Character Canonicalization	Converting characters to a standardized form, typically to prevent “special” characters in user input from being interpreted as control characters by the input processor.
Confidentiality	A guarantee that the secrecy of data is preserved.
Cross-Site Scripting	See XSS.
Cryptographic Agility	The ability to change the solution’s cryptosystem in the face of increasingly advanced attack techniques.
Debug Mode	A method for accessing a system that allows a user to access the application via an alternate interface. Debug modes frequently allow users to inspect and modify internal application state.
Defense in Depth	An attempt to delay or make more difficult the compromise of a system by using multiple defense techniques. Defense in depth techniques are not comprehensive security walls, but make it more difficult or more time-consuming for an attacker to breach the system.
DES	The Data Encryption Standard. A now, partially, obsolete symmetric-key encryption algorithm.

Double Free	A C programming language flaw in which the developer uses the memory de-allocation function free multiple times. A double free may cause memory leaks or lead to arbitrary code execution vulnerabilities.
Entropy	Randomness, for cryptographic purposes this is necessarily unpredictable observing the system or restarting, etc.
Exploit	Software that takes advantage of a vulnerability in order to trigger unspecific behavior.
Fingerprint	Collecting information about a remote system in order to identify it, i.e. to find out what OS, browser, etc. are used. The fingerprint can be used to choose exploits suitable for the remote system.
Forced Browsing	An attack where server resources are accessed by guessing the form of the resource URL.
Heap Overflow	A memory corruption vulnerability that exploits an overflow in the heap, a large pool of memory.
Pointer Obfuscation	An exploit mitigation technique that causes C function pointers to have un-guessable address values, thereby increasing the effort required to create a reliable exploit.
Input Validation	The use of rules, or constraints, to verify the security of data on which an application will operate.
Integrity	A guarantee that data is not modified in an unauthorized manner.
IPsec	The Internet Protocol Security (IPsec) protocol suite allows for secure communications over Internet Protocol (IP) networks. It operates in multiple modes with different security properties.
JTAG	The Joint Test Action Group (JTAG) interface and protocol are used for debugging embedded systems and integrated circuits. Also known as IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture
Key Generation	The process by which a secret, or key, is derived for cryptographic operations.
Network Sniffing	Eavesdropping on network communication.
Output Encoding	Transforming data into a representation that is safe for use in a particular computing context.
Path Canonicalization	Reducing a path from one of multiple ways of representing a target resource to a standardized unambiguous format, that is to say, a direct, well-formed path (without, e.g., elements like "." or "..").
Prepared Statements	A database access technique in which queries are pre-compiled that is efficient and useful for preventing SQL injection. Prepared statements are also known as parameterized queries.
PRNG	A pseudorandom number generator. A PRNG is an algorithm for producing sequences of numbers with properties intended to be indistinguishable from random numbers.
Safe SEH	A platform feature that provides safe structured exception handlers. The use of Safe SEH can be useful in making reliable exploit creation more difficult.

Secure Element	A device capable of providing tamper-resistant, secure application hosting. Frequently used for handling confidential and cryptographic operations.
SQL Injection	An exploitation technique by which attacker submitted data is interpreted as SQL instructions. SQL injection can lead to denial of service, data loss and even system compromise.
SSL	The Secure Sockets Layer (SSL) is a now insecure protocol for performing transport protection. It has been replaced by TLS.
Stack Cookies	A platform provided feature that can assist in the prevention of stack based buffer overflow exploitation. Stack cookies are also known as stack canaries.
Threat	The danger that a computing system may be exploited via some vulnerability.
TLS	Transport Layer Security (TLS) is a secure protocol for performing transport protection. It replaced SSL.
TPM	A Trusted Platform Module is a secure cryptographic coprocessor.
Trust Zone	A set of ARM CPU security extensions that can be used for providing secure element functionality.
URI Parameters	The query data elements of a uniform resource identifier (URI). Most commonly this refers to data elements included in a web application's uniform resource located (URL).
Vulnerability	A weakness that allows an attacker to trigger unspecified behavior.
XSS	Cross Site Scripting (XSS) is a web application vulnerability in which an attacker is able to compel a web browser to interpret attacker supplied data as code. XSS can lead to a complete compromise of an end user's web application account and can even lead to the compromise of an end user's computing device.

5. Appendices

5.1. Appendix A: A Secure Boot Mechanism

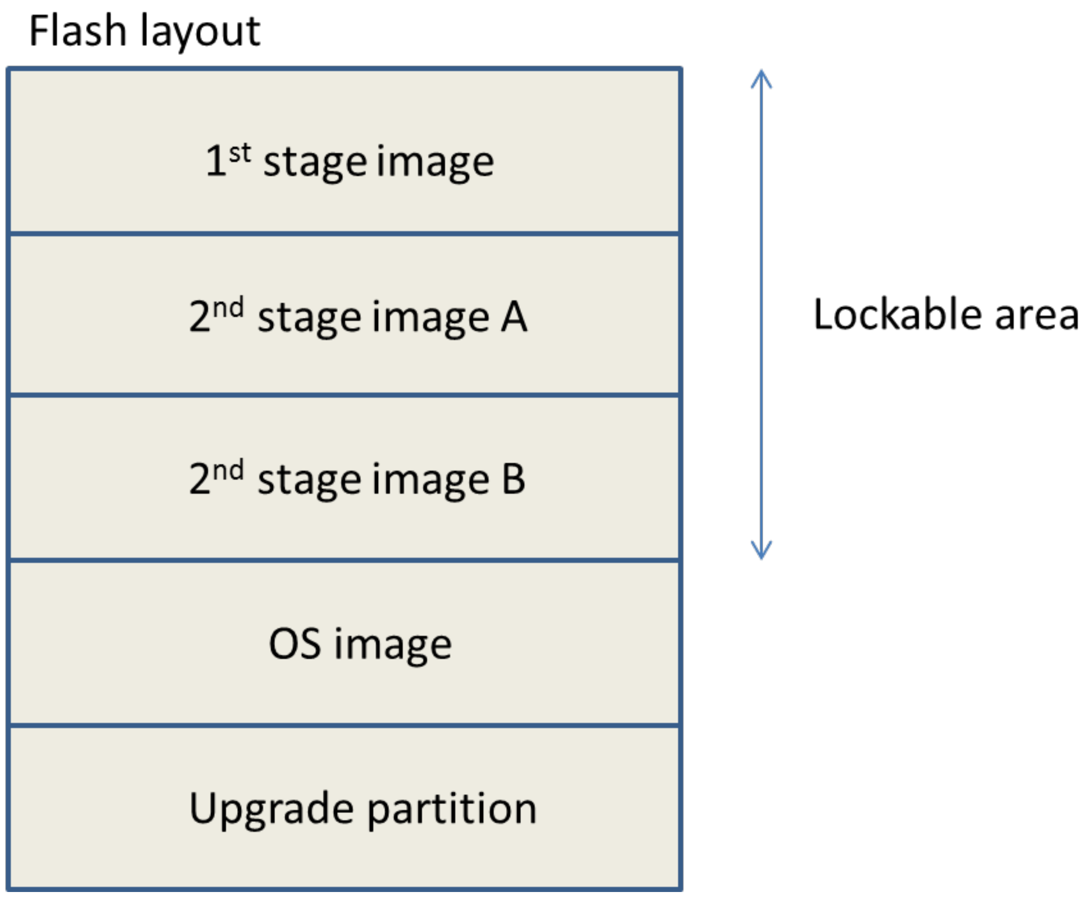
This is a high-level description on how a secure boot and flash design might look. It does not cover the issue of key revocation and a handful of other details. A production implementation must handle key revocation and any details that were omitted from this appendix for brevity.

Introduction

To create a secure boot chain, trust has to be propagated from the bottom all the way up to the running system. This implies that the system has to start from a "known good" point as well as authenticate and verify each step in the chain. T0 accomplish this, the following components are required:

- **Read Only Memory (ROM).** This provides the "known good" start point. From a functionality standpoint, it is used to store code which is capable of finding the next stage, perform a cryptographic hash on the image, compare it to a hash stored in ROM or OTP area and load / execute the image. It is crucial that the code in this stage has been extensively tested for security and stability issues since it can only be upgraded by releasing a new hardware revision.
- **One Time Programmable (OTP) key storage.** This memory area holds public keys / certificates to verify each image. It can also be used to store the hash of the first stage.
- **Lockable Read / Write boot media.** This can be any type of r/w storage media (Flash, Hard drive, etc.), but should allow locking of certain partitions. It must also support access controls of said locks to prevent the OS from unlocking these partitions. This lockable media is used to store 1st and 2nd stages, subsequent stages (including OS images) can be stored on here as well, or on a different media. This media will be referred to as "Flash" throughout the rest of this text.
- **ROM Firmware stage.** A small piece of code committed to a ROM. It is capable of finding the next stage, performing a cryptographic hash of it and comparing that to a known good value.
- **1st Firmware stage.** This stage can locate the next stage as well as cryptographic keys. It is also capable of authenticating the next stage with the help of the bundled cryptographic keys and digital signatures based on asymmetric cryptography. Dividing the ROM and 1st stage portions in two different stages makes the design use less ROM. If the use of ROM is not an issue, this functionality can be folded into the ROM firmware stage.
- **2nd Firmware stage.** This stage is capable of locating and loading the OS image, as well as authenticating the OS image with the help of digital signatures based on asymmetric cryptography. The 2nd stage is also tasked with performing updates and locking the Flash once it transfers control to the OS. The 2nd stage image can contain new keys for next stage to allow for key updates, would that ever be needed.

The reason ROM, 1st stage, and 2nd stage are divided is to leave as little functionality as possible in areas which can't easily be upgraded should there be an issue with the code.



Secure boot

When the system boots, the following procedure should be followed:

1. Boot CPU from image stored in ROM
2. ROM image locates 1st stage within locked portion of the Flash
3. ROM image verifies 1st stage image
    a. ROM image locates 1st stage hash in ROM or OTP
    b. ROM image produces a cryptographic hash over the entire 1st stage image
    c. ROM image compares the stored hash with the computed hash. If they are equal, boot is allowed to continue. If they are not equal, boot is aborted with an error code.
4. ROM image loads 1st stage into RAM
5. ROM transfers execution to 1st stage image
6. 1st stage image locates 2nd stage “last image booted” and “boot successful” values within locked portion of the Flash
7. 1st stage image locates 2nd stage image to be booted with the help of these values
8. 1st stage image verifies 2nd stage image
    a. 1st stage image locates public key / certificate for 2nd stage image in OTP
    b. 1st stage image loads 2nd stage image to RAM
    c. 1st stage image locates the signature for the 2nd stage image within the binary blob loaded into RAM (usually at the start or end of the image)
    d. 1st stage image produces a cryptographic signature over the entire 2nd stage image with the signature in the image set to all zeros
    e. 1st stage image verifies the computed signature with the help of the public key. If they signature is verified, the boot is allowed to continue. If the signature is not verified, the boot is aborted with an error code and the “boot successful” value is set to false.

9. 1st stage image transfers execution to 2nd stage image
10. 2nd stage image verifies if any update is available in Flash update partition. If there is one, the procedure in the next section should be used.
11. 2nd stage image verifies OS image
    a. 2nd stage image locates public key / certificate for OS image in OTP or within itself (to support key updates)
    b. 2nd stage image loads OS image to RAM
    c. 2nd stage image locates the signature for the OS image within the binary blob loaded into RAM (usually at the start or end of the image)
    d. 2nd stage image produces a cryptographic signature over the entire OS image with the signature in the image set to all zeros
    e. 2nd stage image verifies the computed signature with the help of the public key. If they signature is verified, the boot is allowed to continue. If the signature is not verified, the boot is aborted with an error code and the "boot successful" value is set to false.
12. 2nd stage image writes image value to "last image booted" as well as updates the "boot successful" value to "true"
13. Lock Flash partitions holding 1st and 2nd stage images
14. 2nd stage image transfers execution to OS image

Secure updating / flashing

Since the flash will be locked once the system is running, the image on the flash can only be upgraded during the boot procedure. This is to prevent the running OS from having write access to sensitive partitions on the Flash.

1. 2nd stage image is running and has discovered what appears to be an update image in the Flash update partition
2. 2nd stage image verifies update image signature
    a. 2nd stage image locates public key / certificate for update image in OTP or within itself (to support key updates)
    b. 2nd stage image loads update image to RAM
    c. 2nd stage image locates the signature for the update image within the binary blob loaded into RAM (usually at the start or end of the image)
    d. 2nd stage image produces a cryptographic signature over the entire update image with the signature in the image set to all zeros
    e. 2nd stage image verifies the computed signature with the help of the public key. If they signature is verified, the update is allowed to continue. If the signature is not verified, the update is aborted with an error code.
3. 2nd stage verifies that the version of the update image is the same or higher than the version already in Flash
    a. If the version is the same or older, the update is aborted with an error code. This is to help prevent downgrade attacks.
4. 2nd stage image writes the update image to the opposite 2nd stage slot
    a. If the write to Flash is successful, the "last image booted" value is updated to force the new image to be selected on the next boot. If the write is unsuccessful, the update is aborted with an error code and the "last image booted" value is updated to force the old image to be selected on the next boot.
    b. 2nd stage resets the device

About CEA

The Consumer Electronics Association (CEA) is the technology trade association representing the $286 billion U.S. consumer electronics industry. More than 2,000 companies enjoy the benefits of CEA membership, including legislative and regulatory advocacy, market research, technical training and education, industry promotion, standards development and the fostering of business and strategic relationships. CEA also owns and produces the International CES – The Global Stage for Innovation. All profits from CES are reinvested into CEA's industry services. Find CEA online at www.CE.org, www.DeclareInnovation.com and through social media: www.CE.org/social.

About NCC Group

NCC Group is a proven full-service security firm that provides penetration testing, secure systems development, security education and software design verification. Our security assessments leverage our extensive knowledge of current security vulnerabilities, penetration techniques and software development best practices to enable customers to secure their applications against ever-present threats on the Internet. Areas of research interest include application attack and defense, web services, operating system security, privacy, storage network security and malicious application analysis.

About the Authors

Andrew Becherer is a Technical Vice President at NCC Group, an information security firm specializing in application, network, and mobile security. At NCC Group, Andrew specializes in web application/web services security, mobile application security, and client/server testing. Andrew has consulted on the security of an enterprise service bus for one of the nation's largest financial institutions, reviewed the sandboxing features of one of the world's most commonly deployed document readers, evaluated the security of numerous embedded devices and has consulted on the security of industry leading infrastructure, platform and software as a service cloud computing solutions. Andrew was a trainer at Black Hat USA 2008 and a speaker at Black Hat USA 2009 and 2010, a speaker at Amazon ZonCon 2011 and 2012 in addition to a number of other information security venues.

Peter Oehlert is a former Technical Vice President at NCC Group. He has worked in the computer industry for over 15 years, dividing his attention between development and security analysis. He has worked for a handful of security and software companies, including a six year stint at Microsoft in the early 2000's and played a significant role in Microsoft's turn-around on security and development of the SDL. He has done extensive research in the area of fuzzing and dynamic and static analysis.

Consumer Technology Association

Consumer Technology Association Document Improvement Proposal

If in the review or use of this document a potential change is made evident for safety, health or technical reasons, please email your reason/rationale for the recommended change to standards@CE.org.

Consumer Technology Association
Technology & Standards Department
1919 S Eads Street, Arlington, VA 22202
FAX: 703-907-7693 standards@CE.org

Consumer Technology Association™