You've been hacked! Now what?

Haydn Povey
Founder and CTO
Secure Thingz
Cambridge, UK
haydn@securethingz.com

***Abstract* — You've seen the headlines. Whether it's bots infecting home networks, the destruction of industrial systems, or the ability to take remote control of automobiles, the horror stories around Internet of Things security are starting to mount, like bodies in a bad movie.**

The bad guys will keep coming with malicious intent. The attacks on connected devices are only going to get worse and more sophisticated. Hardware, software, communications and communications protocol, device commissioning, applications layers and other systems considerations are just some of the many entry points that could impact security of a device, fall victim to malware, and lead to data breaches or weaponization. Boundary protection can be too porous. Systems that may seem secure today may have weaknesses that will lead to failure in the future. Failure at some point is almost an inevitability.
Privacy, corporate reputations and even lives can depend on the ability to ensure the security of a device.

So it's time to face reality. There's a good chance your device will get hacked. The question is: what are you going to do about it? How will you recover? And what can you do to prepare?

This presentation focuses on what you need to do in the aftermath of an IoT compromise and how you get back to a trusted system.

Keywords—security; hack; secure boot; secure element; IoT; IP; attack; system; architecture; cyber security

I. INTRODUCTION

We are becoming increasingly used to the headlines of hacking across our IT systems. Barely a week goes by where major flaws aren't found in the computer infrastructure that surrounds our digital domain, and whilst these are all concerning attacks, there are two major corrosive effects. First, while the global press may publish details on the attacks, there is a consumer fatigue that "yet another attack has happened," minimising the importance of implementing countermeasures and instilling hygiene amongst users. Secondly, and more egregiously, there is perhaps a hopelessness creeping into organizations, both on what they should do to prevent the impact of an attack on their businesses and how they can increase the security of their own products. While there is little we can do on the first, there is plenty we can do as an industry to solve the second by providing solid frameworks for responding to attacks, and building systems which are intrinsically resilient to these evolving attacks.

The reality for any complex system is that there will always be flaws in their design, implementation, and management. We are all human, and we all have to get systems in market rapidly due to competitive and business pressures. As such we will never have the time or budget to get any system beyond a small kernel that's technically correct and certified by a group of peers. Specifically, we see industry challenges arising in three focused areas: 1) architectural specification; 2) technological inheritance; and, 3) system integration.

A. Architectural Specification

System architectures are the bedrock of technology, and they are often open to group review and public domain investigation. And yet we continue to see many fundamental flaws appearing. We have seen this in the BlueBorne attacks, where flaws in the Bluetooth specification led to multiple attack vectors being discovered many years after definition, and of course we have seen the recent Meltdown and Spectre compromises, where incorrect architectural definition has led to multiple side-channel attacks.

B. Technological Inheritance

We all stand on the shoulders of giants, and leveraging standard hardware and software components is the bedrock of modern computing. However, we are all at risk from compromises within these building blocks, creating a bubbling-up of issues which we are ill prepared to manage. A recent example of this was compromises identified in low-level Transport Layer Security (TLS) communication drivers, which themselves are meant to be highly secure. The nature of these drivers is they are buried deep within the application and are not necessarily easy to patch or manage. In fact, the nature of many communication-level security flaws is that they hold privileged status within the system and present an Achilles heel within the system when they go wrong. In this case, the vendor of the drivers produced a patch when they identified the flaw, and correctly notified the OEMs who had built on this software. However, there is little guarantee that the OEMs and end users had the knowledge or capability to remediate systems in the field.

C. Systems Integration

Similar to technological inheritance, the integration of components from numerous vendors is fraught with issues. It is not always clear how the system will fit together, and if compromises are introduced when systems are built. A classic example of this has occurred in the mobile telephone world where an innocuous incompatibility between baseband chipsets and application processors led to a major compromise in 2017, subsequently patched by the baseband chip vendor. As we build increasingly complex systems, there is a demand that every system must be protected individually, and that the system should implement a "zero trust" model between modules. However, this will necessitate additional cost, and addition effort.

II. DEVELOPING AN INCIDENT RESPONSE PLAN

As mentioned earlier, every complex system will have multiple flaws, and hence we must assume that every system can, and will, become compromised at some point. At some point, you will be hacked. So what are we going to do about it? This paper outlines an initial best practice approach and longer-term mitigation strategy.

Given that being hacked is inevitable, it becomes imperative that every vendor within the value chain understand their role within the industry; that they prepare for having components or systems that become compromised; that they have a pre-planned response mechanism; and that they follow through correctly to ensure the industry's trust in them is maintained if they do not wish to lose brand value. It is also imperative that this incident response plan is not only in place for current products, but that it also covers systems that have already been released. To assist in this, a number of groups, including the Internet of Things Security Foundation (www.iotsecurityfoundation.org), are creating Best Practice Guidelines that IoT vendors can integrate into their own processes.

III. BEING PREPARED

Preparing for the inevitable compromises is a multi-faceted process however the following five components represent a good starting point.:

1. Scope organizational impact
2. Define internal cyber security policy
3. Clear communications policy
4. Develop a bug-bounty policy
5. Execute deep threat analysis

A. Scope Organization Impact

It is difficult to judge how much to invest in protection against abstract threats, when compared against known budget constraints. Hence the first step any organization must judge is what the damage to the organization would be in a worst-case scenario.

For many organizations, this will certainly include brand and reputational damage, but this is being exaggerated by other market forces, including the willingness of the industry to sue for impacts in their supply chain. A vulnerability in a communications stack may leave a process control system, and subsequently a large processing plant, at risk. Hence, a relatively small code fix could potentially protect a massive capital structure. Similarly, where there is the potential for customer data to leak, the balance of additional time and effort to minimise any threats is obvious compared to the €10M minimum fine for a data breach under GDPR regulations.

Best practice: To enable a comprehensive impact, it is often the case that a specialist organization, external to the business unit, should be employed to challenge assumptions and ensure the corner cases are explored. This team may be a central function within large organizations, or may be an external contractor within smaller operations. Generally, this team will need to be supported by internal experts running an insurgency attack to see how they would best compromise their own products.

B. Define Cyber Security Policy

No battle plans ever survive the first encounters of war, but you would never wish to go into battle without one. The same is true of defining a cyber security policy around products for the IoT. We don't know exactly when, where or how the compromises or exploits will be found, but we must have the framework to deal with them.

- Assuming compromises will occur is the first step in defining a policy. We have to expect every device will be attacked, and every device will be compromised at some point in its life. Every complex device containing firmware or software will have exploits. Every device that has a communications stack will have exploits. Every system containing an active microcontroller or microprocessor will have exploits.

  Best practice: Assume all devices will be exploited at some point. Ensure active patch management is possible, and ensure functionality is available within the product to update it. Ensure that patch management is built into the project costs and lifecycle management, and that development tools and firmware can support ongoing releases.

- Executive Ownership is critical in developing a cyber security policy, and responding to an exploit.

However, it also important to have executive overview of security within the products being created. Security must be supported holistically in the organization, not relegated to the IT department.

Best practice: A main board member (CxO) should report fortnightly to the board on any cyber incidences, and on cyber security policy implementation

- Engineering leadership engagement is a key requirement in producing secure IoT devices. Engineering must decide what level of security is required for a product with justification, commit to maintaining software for the lifecycle of the product, and ensure that security test frameworks are part of the signoff criteria

  Best practice: Security must move from an after-thought into the fabric of the engineering process. All architectures, components and sub-components must be validated against an evolving threat model, and existing products should be verified against major new threats.
  Active patch and update management are critical in supporting devices across their lifecycles, with the ability to recover to a known good state.

- Product leadership must continue to own its products over its entire lifecycle, and must integrate security into the fabric of the offering. As such, the transition from selling commodities to services and lifecycle support is incredibly important, but also an important source of additional revenue.

  Best practice: Ensure the product mix includes ongoing lifetime maintenance, or the ability to manage the device to mitigate exploits. To achieve this, low-level security services must be instilled in the device, providing a secure kernel which can be utilised to support ongoing interaction.

- Rapid escalation of issues is critical in gaining support from executives for necessary actions, and to ensuring the industry retains confidence in the organization. This is true whether a flaw has been identified internally, a compromise has been found externally, or an ethical hacker approaches the organization with a new exploit.

  Best practice: Implement a flattened management structure group for exploit investigation and responsiveness.

- Communication policy is critical, and will be covered in more depth later. However, no Best Practice would be complete without a clear communication strategy for when attacks are found, how they are solved in partnership with clients, and how they are communicated publicly.

  Best practice: A set of policies should be implemented ensuring rapid and formalised communications which can convey urgency while demonstrating a managed and resolving situation.

- Proper triage of an incoming compromise is critical to ensure the organization reacts positively without becoming overwhelmed by every single flaw in its products. A formal process of evaluation, investigation, mitigation and communication is required.

  Best practice: A written process for initial engagement with issue is important, including the creation of a template to record all aspects of the incident, ensuring a consistent approach. This enables simple recording and comparison of events, and ensures all stakeholders, including technical teams, suppliers, legal resources and human resources, alongside business management, are educated rapidly.

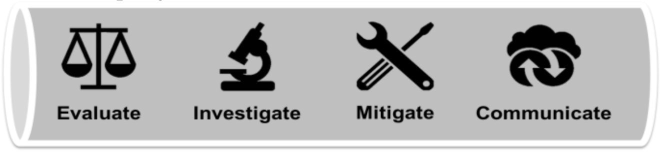

Figure 1. Triage steps.

- Process management should already be key function of the organization. It is important that version management and product evolution are managed aggressively within an organization. This covers both aspects of versioning, where previous versions may have known exploits which have been subsequently fixed. Similarly, product evolution will invariably enable flaws to creep into the system, and there should be the ability to roll back to a known-good version. Identifying where a flaw occurred and how it was introduced are key mechanisms in developing better internal processes to manage compromises over many years.

  Best practice: Maintaining clear records of releases within a mastering system is important in being able to understand when and how flaws were introduced, and who holds ultimate responsibility for these. These records are important from a technical perspective, but also critical from a legal perspective if the organization finds itself in court for a GDPR breach or lawsuit.

- Patching and update management are important outputs from any cyber security policy, as this will be the primary response to any compromise, unless a full recall is required. Ensuring that all systems are manageable requires that low-level services have been fully certified and are small enough that they have no known attack vectors is important to constraining the update process.

Best practice: Ensure development tools support version management and interim releases, and that patches can be signed and encrypted for specific target devices, or product groups. Assume that an update mechanism can also be exploited by an attacker, and therefore provide development frameworks which enable constrained mastering releases of software, with authentication and authorisation. Further ensure that updates are as small as possible to ensure network bandwidth and battery power impact are as small as possible.

- Minimising attack impact through authentication is a clear goal of any cyber security policy. It should be the case that all devices have a strong cryptographic identity associated with them, and that all communications to and from the device have implicit authentication. This way, we should be able to both manage the communications to reduce the attack surface of the device, where an attacker must shape their attack to the unique device; and additionally create a zero-trust framework where devices cannot easily propagate attacks as they do not have authentication capability for other devices.

  Best practice: Ensure all devices have cryptographically strong certificates and authentication mechanisms, including PKI frameworks, the ability to hold secret information securely, and unique addressability, such as X.509 (or equivalent).

C. *Clear communications policy*

As mentioned previously a clear communication policy is the bedrock of a successful Incident Response Plan. However, this is often easier stated than done, especially given the embarrassment and sensitivities traditionally associated with flaws.

There are three stages to a successful incident communications policy

1. Confirmation.
   In many cases a compromise may be identified by an ethical hacker. We will come onto bug-bounties in a moment, however, whether you offer one or not, it is always best to engage positively when approached. In the past, companies often buried their heads in the sand, but today that approach will bring industry condemnation and far greater reputational damage, as it gives the impression that they are both ignoring the issue and disrespecting their customers. Engage positively and strongly, and if possible, attain the use of the hacker to confirm the flaw in as much detail as possible.

2. Notification
   Following initial triage, it is important to notify customers as soon as possible that there is a potential issue and that the organization is working on a fix. If the fix is simple, it may be that the organization will quickly make a patch available when they notify the clients. However, if the fix is complex, it is important to notify clients as early as possible as they will need to clarify how to integrate the workaround into their products. If the issue is catastrophic in nature, it is also important to flag this early to clients to ensure they, and their customers, are aware of any potential impacts to their businesses. A critical flaw in an automotive component may mean vehicles are at risk, and subsequently so are human lives. In this case, a failure to notify will raise the highest legislative impact and could put your company's survival at risk.

3. Publication
   Traditionally potential flaws were seen as embarrassments. Today that approach is changing, and the open publication of flaws and subsequent fixes is a prerequisite for trust within the industry. If you are not publishing flaws, you are seen as trying to hide mistakes, and this itself is a good reason for not doing business with an organization. Publication does not mean self-flagellation, and unless a flaw is critical, you do not have to publish it on the front page of your website. However, there should be a specific publication mechanism, with subscription notification, for users.

D. *Develop a bug-bounty policy*

We have touched on bug-bounty in the communication policy, but it is worthy of additional focus. A company should, as part of its Best Practice, engage with ethical hackers, and we are seeing this encoded into the new laws currently coming through the US Congress, where this practice has been legitimized.

The nature of ethical hacking is that a third-party independently finds flaws and effectively ransoms the information to the organization. This approach is obviously distasteful, although better than a third party finding the flaws and actively exploiting the issue. A better approach is to be aggressive in engaging with the community, to build a predefined set of rules of engagement, and to predetermine where specific value is attributed to finding compromises. This approach also ensures that the bug-bounty explorers are always operating within the law, and they are far more likely to be true white-hats.

Best practice: It is suggested that an engagement framework is published on the organization's website, as part of the Cyber

Response Initiative, and that the executive leader who is tasked with managing product security has a clear responsibility for engaging with this process and paying the hackers. After all, the cost of an exploit going wild is far higher than the cost of managing it internally.

E. Execute deep threat analysis

Of course, all of the above is great for developing a process to manage exploits. However, it is only through running an internal deep threat analysis that the organization will really start to understand how exposed they are.

Best practice: In the first instance, the organization should create an incident response policy and dry run the process. This should involve senior business and technical leadership instigating a tear-down attack on one of their own products, identify all the suspected compromises, and review the potential consequences of a successful attack. In most cases, organizations receive a nasty shock.
Identifying and ranking any exploits forthcoming will help build a set of priorities which engineering can start to work on.

As this approach is targeted at the first set of products, it is likely that common threats emerge, such as communication stacks, lack of prescribed identity, limited patching, and poor version management. These common threats subsequently build a back log of threats to mitigate but also ensure that new products under development learn from these issues, creating a positive engagement across the organization.

Following this initial phase, the policy should be readdressed based on feedback, and then the organization is probably best tasked with bringing in a third-party security analysis organization to review the process and focus on additional areas which may have been missed, or outside of the organizations knowledge base.

IV. RESPONDING TO AN ATTACK

If a full Incident Response Policy has been implemented, then the Response phase to a new attack vector should flow easily. However, each one will be a learning experience and the focus should be on investigation of the attack, identification of the flaw, and ongoing mitigation in both existing and new products through updates and patch mitigation.

The following steps are suggested:

- Identify cyber security incident
  Although obvious, it is imperative that the organization understand quickly whether the compromises found are new issues, or whether these have been found before and are either unpatched exploits or a novel leveraging of existing issues.
  To achieve this the team must be able to replicate the system and rapidly understand the impact and pathology of the attack. This necessitates the use of a "clean room" system disassociated for the main IT system where the attack can be replicated without impacting the company if it propagates.

- Define a clear set of objectives
  The outcome of identifying the pathology of the attack may be many fold.
  First, it may be that the attack is leveraging a known compromise in a new way. This obviously needs to be understood further. However, it may be that the organization needs to warn customers to more rapidly update or apply patches. In many industrial organizations, the adoption of patches is slow as there may be unknown consequences which the client needs to mitigate against.
  Secondly, the attack may be a new variant of a known attack, compromising a new area of the codebase or system. In this case it may be that a patch can be rapidly reworked to cover the new hole, and future products can be coded to mitigate against that attack.
  Thirdly, the attack may be a completely new "zero-day" compromise. In this case the organization needs to rapidly identify the consequences of the attack, and develop a mitigation capability. They will need to judge the impact and potential scale of the exploit, and how quickly to alert their customer base.

- Recover and remediate
  The primary goal of any attack response is to minimise the impact on the user.
  As such, in the first instance the system should be able to enter a quiescent state where advanced functionality is switched off, to reduce the attack surface and inhibit the propagation of any attack. This capability may be a feature of the Real Time Operating Systems (RTOS), or may be reliant on underlying security services, such as a Secure Boot Manager. In the worst case, this functionality may be implemented by a quarantine system within the user, but this should act as a last resort.
  The second phase should be to recover the device to a known good state. For advanced devices, this probably means recovering the Secure Boot Manager, which can be achieved through a soft reset. In this domain, the system should be able to isolate any possible attacks and inhibit them, while holding the main system in a safe-mode or shut down.
  The third phase is to remediate with the release of patches and upgrades which should be applied through the low-level security services, again leveraging the secure boot manager functionality to ensure the patches are signed and encrypted, and to ensure that these are version managed to stop any roll-back attacks.

V. BUILDING RESILIENT SYSTEMS

Previously, we introduced many aspects necessary to recover from an attack, focusing on executing from a known good position, the ability to patch identified compromises, and the desire to update with versioning.

The mechanism to do this within a microcontroller is a Secure Boot Manager, a small and lightweight security kernel operating at the lowest execution level, which is capable of being certified and flexible enough to support the long lifecycles needed for IoT-centric devices.

The Secure Boot Manager, outlined below, is a modular framework which can be configured within a commercial IDE, such as the IAR Embedded Workbench, to deliver a range of solutions that stretch from a very small codebase to a feature rich set of security functions. The Secure Boot Manager itself leverages the device boot framework of security orientated microcontrollers, such as the STMicroelectronics STM32H7 and the Renesas Synergy S5 families, to create a secured domain operating below the RTO, enabling even low-level drivers to be managed an updated over the lifecycle of the devices.

A modern SBM, such as that shown in Figure 2, should implement a rich set of functions to support the next generation of ultra-long-life devices.

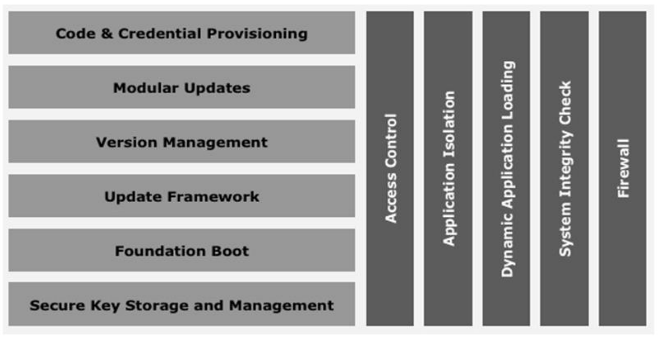

Figure 2. Citadel Edge Secure Boot Manager. (Secure Thingz)

Firstly, it is important that the SBM support the secure key storage and management resident in the microcontroller. Fundamentally, to enable secure services, the devices must have sufficiently secure storage in which to store the private keys upon which the first secure communications over PKI asymmetric cryptography rely. Once a secure channel to the device is established, it is possible to program the device securely.

Through leveraging the secure channel, it is possible to start building the secure foundations a product requires. First, the identity must be provisioned. The identity may be a certificate structure, such as the standard x.509 format, or a more bespoke form tailored to the IoT. The certificate relies on additional private keys being provisioned into the device to lock the device to the certificate. This may be extended with other forms of identity, such as physically unclonable functions (PUFs) or additional ownership keys. Once the device has been provisioned, it can be securely programmed.

Secure programming enables OEMs to develop a master their application, ensuring the code is both signed for specific devices, and encrypted to manage it securely in transit. Through a secure programming system, it is then possible to validate the device, via its certificate infrastructure, and deliver the encrypted image to the device, block by block. The key generated at mastering is also then exposed to the device, unlocking the code and enabling a raw image to be written to flash.

The update framework within the SBM further extends this capability to enable patches generated by the IDE to be targeted to groups of devices or individual devices, based on the identity provisioned into the device, and on the definition of the Security World within the IDE itself. The update can take many forms including a single line code fix, a module, or an entire image depending on the fix being undertaken. In this way we can mitigate the impact of updates to a minimum, and speed the adoption of patch infrastructure.

Additionally, as mentioned earlier in the paper it is important to integrate version management into our IoT devices. Whilst signed images proved they came from a valid source, they may unfortunately contain known exploits, and as such, it is important that attackers cannot explicitly roll-back software to a known-bad version and take control of the device. This solution obviously requires integration into both the target device (MCU) and the development environment to function correctly, alongside the mastering tool which injects and manages keys. Hence a holistic and well integrated system is required.

Modular updates are traditionally outside of the scope of microcontrollers due to the monolithic nature of their memory systems. However, with the advent of the TrustZone for Cortex-M devices (ARMv8-M architecture), we are seeing this change, with multiple modules now laying across protected and open memory. In this context, we can now focus on delivering smaller modules, but obviously the system needs to be compiled and built with this in mind. The advantage to this approach is that we can more easily constrain modules, increasing security, but also ensure that we minimise bandwidth impact alongside conserving battery power, as we do not have to program so much flash.

VI. SUMMARY

The reality of modern connected systems is that every system will be compromised, and as such, the impact on the industry will be long-term and pervasive. The solution to this inherent insecurity is to provide both business and technical frameworks that accept flaws and compromises, embrace them as a way of improving the product; and ultimately drive patches and updates across the lifecycle of the device.

The end perspective is that while we cannot stop flaws we can, and should, continually improve the solution. To achieve this a Secure Boot Manager, delivering low level secure services for patching, updates, remediation and foundational recovery mechanisms are key. The advent of Secure Microcontrollers, leveraging advance technologies such as the ARM TrustZone for Cortex-M, lend themselves to dynamic remediation and recovery, and coupled with advanced Secure Boot Managers, provide the foundation for a secure future.

VII. ACKNOWLEDGMENTS

All trademarks are properties of their respective holders. First published at embedded world 2018.