A Case for a Trustworthy BMC

About

This document was produced by the Cloud Security Industry Summit (CSIS). CSIS is a group of Cloud Service Providers, with a mission to align on a vision and approach to developing best-of-breed security solutions. The group includes members from top Cloud Service Providers, partnering as an industry team and evolving a coordinated approach for improving cloud security from component to system to solution. Intel facilitates the group.

For more information about CSIS's charter and scope, visit www.cloudsecurityindustrysummit.org

We would like to recognize the following contributors to the document:

Yigal Edery, Kameleon
Rob Wood, NCC Group
Tobias Langbein, ZKB
Joseph Reynolds, IBM
Sumeet Kochar, Lenovo
Megha Kalsi, Protiviti
Parth Shukla, Google
Ben Stoltz, Independent engineer
Eric Johnson, AMI
Daniel Moniz, Oracle
Alberto Munoz, Intel
Joerg Heese, 1&1 IONOS

Document Goals

This paper describes the Cloud Security Industry Summit (CSIS) members point of view towards Baseboard Management Controllers (BMC's) security and trustworthiness. We cover some of the background and history of BMC-related security issues, explain why BMCs need to become more secure than they are today, and propose approaches for how to achieve that and get the BMC industry to a point where it is secure enough to become an extension of the system's Root Of Trust (RoT).

We would like to recognize that there are existing industry efforts already in motion to improve on BMC security, many of them started in recent years as a result of negative publicity that brought traditional BMC weaknesses to light. We believe more can be done, and so this paper takes a holistic and comprehensive approach and covers the topic in full breadth.

For impatient readers, we have provided a summary in the form of a Recommendations Checklist that you may skip ahead to.

Introduction & Problem Statement

Servers have been using Baseboard Management Controllers (BMC) almost since the day data centers were born. BMC's provide an essential functionality to data center operations by enabling remote management of servers and their critical functions, without having to physically access the server. This is essential to operate systems at scale.

Historically, Original Equipment Manufacturers (OEM) provided their own custom BMC solution in the OEM server market. A classic BMC consists of a System-on-Chip (SoC), which is an independent microcontroller and a software stack that provides the remote management functionality. The software stack typically comprises some flavour of Linux operating system and several BMC applications running on top.

The core functionality of a BMC is to enable remote management and monitoring of a server, traditionally done over the IPMI protocol. Sample functions include the enablement of IPMI based commands over LAN or system local interface, remote provisioning, remote interactive management via a virtual keyboard, video and mouse (KVM), log monitoring and notifications, and sensor-based data collection and alerts for things like thermal management.

Each function typically has an associated IPMI command which allows administrators the ability to automate their work and utilize additional network services that read and write via IPMI. Data centers are growing in size and the resources needed to manage large server farms dictates a need for automation to enable them to scale. BMC functions offer companies and administrators the ability to scale via automation and more effectively manage hardware resources. In addition to standardized IPMI protocol, it is common for vendors to implement vendor specific IPMI commands or other mechanisms to administer systems and manage the BMCs themselves.

Knowns Concerns

BMCs, by virtue of their nature, are able to execute a large variety of privileged operations on the respective system. They present a perfect avenue to achieve various goals that attackers want to pursue. These goals can be grouped in the following categories.

1. **Attacking the BMC from the host system:** An attacker who compromised the host system could use their access to also compromise the BMC. Ways to abuse this compromise would be to reflash BMC firmware or install BMC rootkits / malware thereby achieving persistence irrespective of the state of the operating system (such as in the case of the "pantsdown" vulnerability[1]). A way to achieve that could be to maliciously re-flash the BMC firmware through the host. Such persistence usually goes unnoticed by typical host security solutions.

2. **Attacking the host system from the BMC:** An attacker with access to the BMC can use the remote management facilities to subvert the host system. An example for this has been USBAnywhere[2], which could be used to mount virtually any USB device to a remote server. Other ways to attack the host include launching permanent DoS attacks (e.g. via thermal management) or modify system run-time behavior (via debugging interfaces).

3. **Attacking the BMC directly:** The BMC exposes interfaces, such as IPMI, over the network. These interfaces can be used to attack & exploit the BMC and gain control over it. Another form of direct attack is through physical presence and physical interfaces (such as JTAG, physical replacement of the BMC boot Flash, etc). These are especially risky when systems are used in locations where physical security cannot be assured, such as when hosted at 3rd party facilities or in Edge points of presence. Regardless of how an attacker gained control over the BMC, they now can perform the attacks mentioned in bullet 2 above.

The susceptibility to attacks of BMCs is rooted in a couple of core aspects that are described hereafter.

1. **BMCs have a perfect vantage point:** As Trammell Hudson pointed out in his Modchips of the State talk at the 35th Chaos Communication Congress in 2018[3], the BMC often has access to the host firmware via SPI (serial peripheral interface) and to host memory through DMA (direct memory access). The BMC gets DMA access because it is on the PCIe (peripheral component interconnect express) bus as a device. This means it can inject code into the host's firmware and memory, and even compromise the operating system.

2. **Not all BMCs have been designed with security in mind:** Most of those attacks stem from the fact that BMCs were originally designed with a threat model that is not suitable for the cloud era, and as trusted proxies for physical presence. Contrary to their privileged nature they are not

[1] https://www.flamingspork.com/blog/2019/01/23/cve-2019-6260-gaining-control-of-bmc-from-the-host-processor/
[2] https://eclypsium.com/2019/09/03/usbanywhere-bmc-vulnerability-opens-servers-to-remote-attack/
[3] https://queue.acm.org/detail.cfm?id=3378404, https://trmm.net/Modchips#Defenses

particularly hardened with defense-in-depth in mind from their inception throughout the various design, integration and implementation stages. IPMI as a standard management interface, with its' long history of public vulnerabilities, is a perfect example[4].

3. **BMCs implement a lot of functionality with a large attack surface:** Popular OEM BMCs offer a large variety of functions and a similarly broad attack surface that has not been reduced to the ultimate minimum. BMCs provide attack surface even on the hardware level[5]. Some BMCs - to this date - are configured using insecure default credentials and protocols. Other examples include missing input validation that repeatedly allows for basic CSRF (Cross site request forgery) or XSS[6] (Cross site scripting) attacks to easily be carried out. Another common class of attacks are buffer overflows, for example.

4. **BMCs most of the time do not have boot time integrity:** Functionalities like secure boot and attestation are vitally important to ascertain the integrity of the BMC's firmware. However, this is far from being used in a standardized manner. It is worth noting that some vendors are making advancements on this front, and modern BMCs are starting to incorporate proper secure boot mechanisms.

The Core Requirements

Thus there are a few core requirements to be met by BMCs, in highly demanding and challenging environments such as those of cloud providers, in order to increase their trustworthiness and resilience to cyber attacks:

1. **Implement highly secure code** - Follow secure development best practices[7] & fully scrutinize the BMC code to eliminate as many potential bugs as possible.
2. **Minimize the attack surface** - Eliminate or make optional unnecessary functionality and thus minimize the TCB.
3. **Protect against execution of unauthorized firmware** - via mechanisms such as Secure Boot and attestation.
4. **Decouple SoC hardware from BMC software stack** - Enable cloud providers to separately choose the best of breed SoCs, and best of breed BMC stacks. Decouple the ability to provide software updates from dependency on hardware vendor support.
5. **Secure the interaction with the BMC -** make the interactions with the BMC (e.g. via physical interfaces, network interfaces, protocols) secure by design, authenticated and subject to authorization policies.
6. **Enable multiple trust domains -** Inside the BMC, not all code needs equal permissions. Embrace the principle of least privileges and assign these on a per-service or per-user of the BMC. For

[4] https://threatpost.com/vulnerabilities-in-ipmi-protocol-have-long-shelf-life/106480/
[5] https://www.nccgroup.com/us/about-us/newsroom-and-events/blog/2018/october/much-ado-about-hardware-implants/
[6] https://www.tenable.com/plugins/nessus/124119
[7] https://github.com/opencomputeproject/Security/blob/master/SecureFirmwareDevelopmentBestPractices.md

example, in a cloud provider bare-metal scenario, the BMC could have privileged operator-facing functions vs. less privileged tenant-exposed functionality.

Anatomy of a BMC

Before diving into proposed approaches of how to address the core requirements in BMCs, we should first take a deeper look into how BMCs are constructed. A BMC is not one monolithic thing, but rather it is built of four layers, each of them would benefit from different approaches to how to secure it. This chapter discusses these layers.

1. The Hardware Layer
2. Pre-Boot Firmware
3. The Operating System Layer
4. BMC Applications & Services

Each of these layers has an impact on the overall security and trustworthiness of the BMC, and decisions made when selecting the SoC as well as the board design itself can impact the trustworthiness of the entire platform.

The Hardware Layer

The BMC SoC typically supports a myriad of busses in order to provide system designers (i.e. OEM/ODMs) with the most capabilities and features possible. A typical SoC will have the capability to provide remote Keyboard, Video & Mouse (KVM) access for the host system as well as access the PCIe, SPI, I^2C, USB, LPC and other busses. The host system can typically access the BMC through one or more of these busses. This means that security flaws in the hardware implementation or in the configuration of the SoC could potentially allow the host system to attack and compromise the BMC, and vice versa.

The hardware therefore must be able to support a cryptographically verified secure boot, anchored in ROM with a hardware-protected public key, to ensure the integrity of the BMC firmware being loaded. Additionally, it must be possible to permanently disable unused hardware interfaces and functionality blocks. These are discussed in detail in the next chapter as part of the secure boot and modularity requirements.

For deployments in locations with reduced security (e.g. third party hosters), edge points of presence, or other untrusted locations, it is vital that the hardware be free of programmatic security bypass mechanisms such as override pins (that could be connected to accessible jumpers & dip switches), and unauthenticated debug ports.

Pre-Boot Firmware

BMC firmware typically uses boot loaders to generically load their embedded operating system. The

boot loaders, in turn, need to maintain a chain of trust starting from power-on/reset until the BMC OS is loaded. Most existing BMC software rely on Das U-Boot[8] as both the first and second stage bootloader. Since 2013 U-Boot has supported 'Verified Boot' which is similar in nature to UEFI secure boot. In this way it is possible for the bootloader to ensure the integrity and authenticity of the OS that is packaged as part of the BMC firmware.

However, the verified boot functionality can only be relied upon if the SoC has the ability to measure and verify the integrity of U-Boot prior to turning control over to it. U-Boot also contains a rich set of tools for debugging and developing the bootloader for new platform configurations.

These powerful capabilities must be properly configured in production systems to ensure that the verified boot procedure cannot be tampered with at boot time, and public tools are available to assist with this[9]. It is also important to ensure that the cryptographic algorithms used to perform the verified boot are expected to be secure for the life of the hardware – often a period of 8+ years.

The Operating System Layer

Modern day BMCs typically run an embedded Linux distro to provide the needed apps and services used to manage the host system. The use of Linux increases the richness and depth of features that may be added to the BMC, provides the firmware developers with access to security fixes and best practices from OSS, but comes with the tradeoff of increased attack surface of the entire stack, especially when not configured with security in mind.

The security benefits provided by Linux require that the firmware developer properly implement best practices, such as attack surface minimization, separation of privileges (create users for services / remote admin users), read only filesystems for all but BMC configuration, SELinux, regular security patching, etc.

Such practices evolve over time as vulnerabilities are discovered, and new attacker techniques and tools are developed so the threat model and mitigations for the OS layer of the BMC must be updated from time-to-time to ensure that these best practices are kept properly aligned with future threats and security features.

BMC Applications & Services

The BMC provides a variety of services, some of which overlap with each other (IPMI versus Redfish, for example). These services accept untrusted input and interact with privileged resources. Since the concept of the BMC is many decades old, not all of the services and capabilities of the BMC were designed with the security requirements now present in the datacenter.

This is where most of the vulnerabilities exist, and recommendations such as disabling unnecessary

[8] https://en.wikipedia.org/wiki/Das_U-Boot
[9] https://github.com/nccgroup/depthcharge

interfaces, enabling secure configuration, and generally following firmware development best practices come into play. The next section covers these in depth.

Recommended Approaches

As discussed in the introduction, remote management functionality is critically needed, but the security concerns are also non-trivial. Also, the BMC is actually built from multiple layers, and different approaches could be applied to different layers of the BMC "stack". We suggest that the following approaches, applied correctly on the different layers of BMCs, should be used to mitigate these BMC security concerns.

Implement Secure Development Practices

Security starts with designing and developing with the right set of security best practices, and following a secure development lifecycle. This aspect has been deeply addressed in a previous work by CSIS, and we consider that a critically necessary approach that has to be applied on BMC stacks (and on any code that needs to run in a cloud environment, for that matter).

For the detailed set of recommendations and guidelines, including some that are specific to BMC's, please review the Secure Firmware Development Best Practices, a document created by CSIS members and then contributed to the Open Compute Project (OCP) and adopted by the OCP-Security[10] workgroup.

Using Modularity to Reduce the BMC footprint

A key principle in security is to reduce the attack surface by minimizing the amount of code that needs to be trusted to not contain hidden/unknown security issues, and reducing the complexity of exposed interfaces.

BMCs targeted at general purpose use, have evolved to contain more functionality than actually needed by many cloud providers. Some of this functionality is derived from historical needs, and some is simply addressed by cloud providers in different ways. For example, remote KVM functionality is hardly ever used at scale, because it assumes an admin manually connecting to a server and configuring it via a Graphical User Interface. Instead, a serial console or equivalent is often preferred by hyper-scale cloud providers.

We believe unnecessary functionality should be removed whenever possible, or made flexible for cloud providers to opt out from. Opting out means the code should not be deployed at all (rather than being disabled and dormant). BMCs should be made modular, and there needs to be an ability for cloud

[10] https://www.opencompute.org/wiki/Security

providers to build and deploy only the subset of functionality they actually need.

Modularity should be applied at “build time” and not at run time. Code that is deemed by the cloud provider as not necessary should not be compiled and deployed at all. Disabling code at run time usually implies there is a way (either legitimately or maliciously) to re-enable that code, which introduces an unwanted attack vector. One can look at disabled, or "dead" code, as a library of functionality only useful to an attacker to use through return oriented programming (ROP), glitch, or other methods.

Reducing Silicon Attack Surface

At silicon level modularity is harder because there’s no “compile-time” opportunity to leave parts out. Still, BMCs should offer the ability to make some silicon blocks optional via OTP fuses, ability to disable some parts after boot if they are only required during power up, or even create smaller-footprint silicon SKUs optimized for cloud providers' needs.

We believe there’s a need to rethink BMCs hardware and create options without the functionality that is not required at scale (such as KVM and virtual media or USB ports which introduce their own unique physical attack surface).

Creating smaller BMCs has the side benefit that it may also help reduce cost, which is another important consideration for cloud providers.

Support Secure Boot

BMCs are notorious for historically not having a proper secure boot mechanism. This has led several cloud providers to actually implement a separate chip that provides external verification of the BMC image prior to allowing the BMC to load[11]. These solutions, while necessary, are less ideal. Cloud providers would still prefer having proper secure boot mechanisms implemented in the BMCs themselves.

For BMCs to be considered trustworthy, they must properly implement Secure Boot best-practices. We recommend looking into Open Compute Security set of specs[12] and implementing the relevant functionality intrinsic to the BMC at both the hardware level (SoC) and the operating system layer (The Linux running on the BMC).

One important case for cloud providers is how to handle secure boot failures (e.g. due to failed verification of the integrity of the firmware being loaded). In a cloud environment, where everything is automated and physical access is the last resort, devices (BMCs included) should never simply fail and halt at boot. They should signal the failure in a secure way and be ready to report their bad state to

[11] E.g. Google’s Titan, Microsoft’s Cerberus and AWS’s Nitro.
[12] https://www.opencompute.org/wiki/Security

attestation requests and be recovered by the cloud provider. For more details, please refer to the "Device Recovery" section in the OCP SecureBoot Spec[13].

Support Secure Configuration

Secure Boot focuses on code integrity of the BMC, but keeping a secure configuration is equally critical, especially given the very powerful position the BMC has in a system. A BMC can be used to re-flash a system, or even to watch & modify code running in the system. Some of these advanced capabilities could be used for the wrong purpose if mis-configured.

A few examples of common pitfalls that lead to non-secure configuration include:

- Complex configuration parsing - parsing is frequently error-prone and buggy which can enable attackers to gain access by manipulating configuration values to exploit command injection or memory corruption issues. If the configurations were signed, only an authorized user could reach that attack surface.
- Some configuration settings can disable security settings like HTTPS and KVM encryption, or enable weak IPMI modes, weakening the security of the system.
- Settings control logging functionality, potentially allowing an attacker to send sensitive log contents to a malicious host.

To mitigate this risk, BMC configuration formats should be kept as simple as possible, protected for integrity and not modifiable unless signed and authorized by the operator. Configuration should be treated as sensitive, just like code, from a security attack surface perspective.

Securing the BMC's interactions

Interactions with the BMC generally happen in one of two ways: Either over a host interface or via the network. Irrespective of the interface used, we are calling out the following generic properties that need to be ascertained in order to secure those interactions.

- **Authentication, Authorization, and Audit logs**
  Actions to be carried out need to be authenticated and authorized on all interfaces. Proper privileges need to be assured. Actions that have been carried out need to be securely recorded in auditable logs[14]. To secure network based authentications, strong automatable schemes such as mTLS or OAuth need to be supported.

- **Input validation**
  Data consumed by the BMC needs to be thoroughly validated irrespective of the interface. This is a generic issue, but it has been known to be exploited in the context of BMCs so we're calling

[13] https://docs.google.com/document/d/1Se1Dd-raIZhl_xV3MnECeuu_I0nF-keg4kqXyK4k4Wc/edit#heading=h.86ysm9rdn308
[14] https://csrc.nist.gov/publications/detail/sp/800-92/final

it explicitly again in this context. Additional guidance can be found in the CSIS Secure Firmware Development Best Practices, which was mentioned previously in this document.

- **Transport security**
  Assurance of the authenticity, integrity, and sometimes confidentiality of transmissions need to be confirmed through signing and encryption. For network interfaces, TLS v1.2+ should be considered as a minimum standard. Certificates used by the BMC need to be protected by secure trust stores. Great care has to be taken to use certificates from a trustworthy PKI[15]. In addition, for host interfaces, the host should not be considered trusted, and must not be considered a single privilege domain. It has to be possible to isolate the network traffic of the BMC (e.g. VLAN, dedicated NIC). The ROM of the NIC the BMC uses needs to be protected from untrustworthy hosts and therefore be cryptographically validated (e.g. dedicated option ROM or attested by the host).

- **Configurability**
  It is understood that BMCs need to support a large variety of interfaces in order to allow for customization to individual requirements and use cases. However, it must be possible to disable any hardware or software interface that can't adhere to the requirements listed in this section, in order to reduce the attack surface. This is particularly true for some management protocols (e.g. IPMI), that may not be able to adhere to the rest of the requirements.
  Protocols that should be supported for security reasons by default include: Redfish, PLDM over MCTP.

Attestation and Measured Boot Support

Attestation is the process of cryptographically proving a system integrity to some external verifier. In this context, attestation is tightly related to having support for Measured Boot. In a typical cloud environment, platforms are expected to attest for their integrity to the cloud fabric, before being admitted into it.

To enable this capability, BMCs need to provide measurements of their boot firmware, configuration and flow in order to allow them to be authenticated. This could be done using the TPM in the system, or in some other mechanism (e.g. by updating BMC designs to be able to provide secure measurements). As of today, most BMC stacks do not have that ability, though there are some early efforts happening in that direction.

In order to attest, the boot firmware and configuration of the BMC needs to be measured, as well as the operating system and the applications loaded on the BMC. These measurements need to be accessible, in a trustworthy way (i.e. signed with a trusted certificate), to the platform verifier.

[15] https://csrc.nist.gov/publications/detail/sp/1800-16/final

Note that this is separate from the possibility to use the BMC as a verifier inside a platform and perform peripheral authentication based on the OCP-security attestation spec. A BMC should not be trusted to be the verifier if it can't prove its own integrity.

Also note that historically, there have been several cloud-provider home-grown RoT solutions (e.g. Titan, Cerberus, OpenTitan, Nitro security chip), each implemented their own attestation. Open Compute is where most of these efforts are now converging to a standard, leveraging DMTF protocols such as MCTP and SPDM. BMCs should adopt the OCP approach for attestation and support that direction.

The Recommendations Checklist

This section provides a summary of the recommended approaches, mapped to the core requirements, and applicability to the four layers of the BMC stack, as an easy to follow checklist.

Requirements and Recommendations	SoC HW	Pre-Boot	OS	BMC Apps
Implement highly secure code				
❏ Follow Secure Firmware Development Best Practices	Y (ROM Code)	Y	Y	Y
Minimize the Attack Surface				
❏ Implement build-time flags to enable compiling a minimal subset of BMC software	N/A	N/A	Y	Y
❏ Deploy only the required subset of OS components	N/A	Y	Y	N/A
❏ BMC SoC should have a mechanism to disable unused silicon blocks in a way that can't be re-enabled without authorization.	Y	N/A	N/A	N/A
❏ Document all the relevant modularity options and their security implications, to help customers select the right choices.	Y	Y	Y	Y
❏ Do not implement any hard-coded backdoors & bypass flags, as these can be used to defeat the authorizations set by the operator.	Y	Y	Y	Y
Protect against execution of unauthorized firmware				
❏ Implement Secure Boot (verifying all code and configuration against valid signatures), rooted in	Y	Y	Y	Y

hardware and all the way up the stack. Recommended: Follow the OCP-Security spec for Secure Boot.				
❑ Implement support for measured boot, starting from the trust anchor in hardware, and all the way up the stack. Recommended: Follow the OCP-Security spec for attestation.	Y	Y	Y	Y
❑ Implement signing of configuration data (not just code), and make all critical configuration changes pass signature checks / authorization checks.	Y	Y	Y	Y
Decouple SoC hardware from BMC software stack				
❑ BMC operating system and apps should have a hardware abstraction layer, to enable easy adoption to different SoC's	N/A	N/A	Y	Y
❑ BMC SoC should not be hard-coded to a specific software stack.	Y	Y	N/A	N/A
Secure the interaction with the BMC				
❑ BMC should allow for configurability of interface support	Y	Y	Y	Y
❑ BMC should authenticate, authorize and enable secure audit of all actions	N	Y	Y	Y
❑ BMC should validate all input irrespective of the interface	Y	Y	Y	Y
❑ BMC needs proper transport security for all network-based transmissions	N	Y	Y	Y
Enable multiple trust domains				
❑ BMC applications and services must follow the principle of least privileges (not run as "root", assigned only the permissions they need).	N/A	N/A	N/A	Y
❑ Operating Systems used on the BMC should support privilege separation by process/service, including access control authorization to hardware resources. (e.g. Not all BMC apps need permission to re-flash the BIOS)	N/A	N/A	Y	N/A

Existing Industry Efforts & Trends

This section includes a breakdown of CSIS's specific point of view and feedback towards some notable industry efforts that are in motion around improving BMC, or more broadly around how to securely manage server platforms. This is not an exhaustive view of all existing efforts, but rather offers the CSIS point of view on two important initiatives.

Open Compute's DC-SCM

Open Compute foundation is engaged in an initiative to standardize a management & security module. The control module is called "DC-SCM" and the interface that enables it is referred to as "DC-SCI".[16] (Note that there is an earlier initiative that has been around for a while called "RunBMC", which could be seen as a subset, moving the BMC card to a standard PCIe slot).

The approach taken by the DC-SCM project offers the ability to decouple security & management from the platform itself, which is great at addressing several of the core requirements mentioned in this doc. The BMC/DC-SCM may become the source of the initial main board boot firmware, allowing removal of the SPI boot-flash from the main board. This is also important when main-boards are decommissioned from a secure data center as the SPI flash should be considered equivalent to any HDD w.r.t. security considerations. In addition, portable secure BMCs have potentially longer life than main boards, which can help reduce costs and improve security.

The DC-SCM effort is still evolving, and interfaces and specs are still work in progress, so we want to call out several worries that we believe should be addressed as part of this effort:

1. **Will it achieve true "freedom of choice"?** Current design seems to focus on pin definition and standardizing the physical interface, but to achieve true ability to pick and choose, there needs to be a clear definition of the semantics on the interface. For example, will a DC-SCM card designed for servers from one OEM fit servers from other OEM? Will all the thermal management IO pins have the same semantic meaning? Will the mux/hub used between the mainboard and the SCM card to route all BMC I/O's be well-defined for interop? Will there be a set of common "BMC policies" that are applicable across platforms? All of these are needed to truly be able to match any SCM card to any platform using the standard interface.
2. **Does it achieve true modularity?** Will it be possible to decouple the hardware part of the stack from the firmware / choice of operating system / BMC applications? Will it be possible to select BMC software stack separate from the HW SoC used on the SCM card?
3. **Can SCM be the new main RoT for the platform?** By design, it has a RoT, but is that meant to be the new platform master RoT, or is there still a need for additional RoT's on the platform for components that are independently managing their power-on sequence? And is the fact that

[16] https://www.opencompute.org/wiki/Server/DC-SCM

the SCM card is detachable open up a new security threats? How does one strongly bind the SCM card to the specific platform it is supposed to protect?

4. **Does it achieve improved code security?** SCM adds RoT on the SCM card, so it helps achieve BMC firmware integrity, but how does SCM address the requirements to harden / improve the security of the code quality of the BMC stack? How does it help reduce the total TCB of the management stack? Does it open a new attack surface by making the RoT easily detachable from the platform it is meant to protect? Are these goals in scope for this project?

We believe these are core questions for the team defining DC-SCM to consider and address as part of the final V1.0 design.

OpenBMC

The OpenBMC project is "a Linux distribution for embedded devices that have a BMC" (per https://github.com/openbmc/openbmc). Below is our analysis of how it addresses various elements of this paper.

We want to first recognize that the OpenBMC project does employ good security practices and defense in depth overall. The project appears to be progressing to meet most if not all of the requirements listed in this document.

Specific examples for such security mindset includes:

- A set of best practices for the development lifecycle and new code contributions[17] including static code analysis on some repositories (e.g. the BMCWeb Redfish server)
- A security working group that meets regularly[18] to constantly push for improved security.
- A security incident response team is established to give the project time to address security problems before public disclosure[19].
- The release process has a reminder to review the security aspects of the release[20] [21]
- Digital signatures are created over the firmware image and are validated during the firmware update process.

In addition, several security-focused features are worth mentioning:

- There is a work-in-progress design to implement BMC secure boot and U-Boot verified boot[22].

[17] https://github.com/openbmc/docs/blob/master/CONTRIBUTING.md
[18] https://github.com/openbmc/openbmc/wiki/Security-working-group
[19] https://github.com/openbmc/docs/blob/master/security/obmc-security-response-team.md
[20] https://github.com/openbmc/docs/blob/master/release/release-process.md#freeze

[21] https://github.com/openbmc/openbmc/wiki/Security-working-group#security-end-of-release-checklist
[22] https://gerrit.openbmc-project.xyz/c/openbmc/docs/+/26169

- The BMC's root file system follows the principle of least privileges. It is mounted read-only, has read/write overlays (overlayfs) for directories like /tmp and /etc, and uses a separate mount for /var. This may make it harder for an attacker to make their intrusion permanent.
- There is an intention to isolate security domains within the BMC. For example, to run processes with minimum authority needed[23].

OpenBMC also addresses modularity, decoupling from hardware, and static (build-time) configuration by taking advantage of Yocto/OpenEmbedded project features[24]. Specifically:

- Board Support Packages (BSP[25]) address decoupling from a specific SoC. OpenBMC supports several SoCs. E.g. the ASpeed BSP is here https://github.com/openbmc/meta-aspeed.
- Device tree bindings (DTB) address decoupling from a specific host architecture[26].
- BitBake layers and recipes allow for easy build (compile) time configuration of which pieces are present in the image. For example, you can configure host console access out of your image using a single line in your BitBake recipe.

OpenBMC supports enabling and disabling BMC interfaces (like SSH and IPMI).

- All of the interfaces are nicely documented[27] which is useful in order to understand the attack surface and strategize on what to deploy/enable.
- Entire services and their interfaces can be compiled out of the image by configuring simple bitbake recipes as explained above.
- Services can be stopped and disabled by using the `systemctl` command. The OpenBMC project is working toward supporting the Redfish NetworkManagerInterface APIs.
- User accounts can individually be allowed access to each BMC interface per OpenBMC User Management Group Roles[28].

On the securing the interface front, OpenBMC has initiatives in motion to replace IPMI with better and more secure alternatives:

- OpenBMC implements portions of the Redfish spec[29] which is intended to replace network IPMI. The implementation is embedded in BMCWeb, the custom HTTP/Web server.

---

[23] https://github.com/openbmc/openbmc/issues/3383

[24] General guidance on the topic here: https://www.nccgroup.com/globalassets/our-research/us/whitepapers/2018/improving-embedded-linux-security-yocto3.pdf

[25] https://www.yoctoproject.org/docs/1.8/bsp-guide/bsp-guide.htm

[26] https://github.com/openbmc/openbmc/blob/master/poky/meta/classes/kernel-devicetree.bbclass

[27] https://github.com/openbmc/docs/blob/master/architecture/interface-overview.md

[28] https://github.com/openbmc/docs/blob/master/architecture/user-management.md#supported-group-roles

[29] https://github.com/openbmc/bmcweb/blob/master/Redfish.md

- OpenBMC has a basic implementation of PLDM[30] over MCTP[31]. These are intended to replace host IPMI.

At the same time, OpenBMC still supports IPMI, including out-of-band network IPMI operated by the `ipmitool` command, and host-facing IPMI. OpenBMC does have plans to strengthen network IPMI security[32], but we believe it's better to follow the recommendations mentioned in the previous CSIS document (Secure Firmware Development Best Practices) and the IPMI promoters own call for action[33], to deprecate IPMI and replace it with newer protocols designed with security in mind.

From the CSIS's point of view, the project is heading in a very good direction. We would love to see more focus on security, including more security focus in the contributions page, educate and emphasize the security mindset among the OpenBMC community, come up with security-oriented guides on how to configure OpenBMC in the most secure way, and prioritize with a sense of urgency some of the efforts already in motion, such as the support for Secure Boot and measured boot mentioned above, the transition to RedFish, and the support for trust domains for BMC applications.

We would also like to encourage platform vendors to leverage the goodness of OpenBMC and support it, and to carefully use the modularity and configurability options of OpenBMC and ship systems that are configured to be secure by default.

Opportunities

Combining BMC's and Platform RoT's

There are initiatives that suggest that eventually, BMCs should assume the role of being the platform's Root of Trust. For example, within the OCP-security group, it is expected that BMCs could take the role of being the platform verifiers for attestation of peripherals.

We believe that with a trustworthy BMC, this definitely becomes a viable option. BMCs are more powerful than current generation of platform RoTs, and have the technical ability to perform some RoT functionalities, such as peripheral attestation.

However, this should become a platform architectural choice. There are good reasons to keep the RoT separate. For example, providing an identity to the platform, acting as a first line of defense against firmware persistence attacks, being replaceable separate from the management component, providing RoT consistency with non-server platforms, or allowing custom RoT solutions that offer more functionality than BMCs will offer.

---

[30] https://github.com/openbmc/pldm
[31] https://github.com/openbmc/libmctp
[32] https://gerrit.openbmc-project.xyz/c/openbmc/docs/+/31548
[33] https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-home.html

At the same time, if BMCs are truly trustworthy, then there are cost and 'real estate' savings to be had by consolidating. Note, however, that while there may be benefits at the platform level to having a single RoT, it remains important for each chip/SoC to have its own immutable trust anchor to avoid common vulnerabilities to physical attacks.[34]

Last, in a platform where both a RoT and a BMC exists as separate entities, there will be a need to define the boundaries and trust relationships between them. The industry will need to define what functions are better left for a discrete RoT to perform, and which security functions should be handed over for the BMC to handle.

To summarize, we see consolidation of BMCs and RoT as a viable direction, but for that to happen, BMCs should become much more trustworthy than they have been till now, and even then, for some platforms, there may still be valid business reasons to keep them separated.

Enhancing Platform Security with the BMC

BMCs were originally designed with a threat model that, for the most part, no longer applies in modern data centers or satellite systems. Throughout this document, we identified the required BMC improvements necessary for our times and for dealing with the modern threats landscape. Once BMCs become trustworthy, it opens the door for more than just consolidating with the Root of Trust. There are major opportunities with having a next generation of BMCs that are a trusted part of the platform. The vantage point of the BMCs, which attackers love so much, could actually become a strength for defenders and for improving the platform security!

In truly trustworthy BMCs, we can envision some new security use-cases which today could also be imagined and hoped for. For example:

- **Act as the trusted agent for external attestation.** The BMC has a dedicated network interface, usually routed over secure management networks. A trustworthy BMC could act as the local entity verifying and attesting local devices (over MCTP), and communicating the platform attestation status to the cloud operator management platforms, without having to go via potentially compromised hosts.
- **Centralized security policies -** The BMC could become the processor that applies local security policies on the platform. This can be hard to achieve in a horizontal world of many different platform configurations, and will require some standardization of a platform manifest, but it could be very powerful if BMCs had a well-defined view of what a platform is, and enforce flexible, cloud-provider controlled policies, on all systems regardless of manufacturer.
- **Verifying "dumb" peripherals** - Some peripherals are always going to be "dumb" and not support advanced attestation. BMCs could take the role of verifying them and attesting on their behalf. Even for smarter peripherals, if the BMC has access to verify them can help reduce the need for a dedicated silicon RoT on such peripherals.
- **Enabler for "Bare Metal" scenarios.** BMCs can be the anchor of trust for cloud providers, even

[34] https://www.platformsecuritysummit.com/2019/speaker/wood/

when a server is rented out to tenants. In such a model, BMCs could provide the foundation for secure transition of system ownership between cloud providers and tenants wanting to "rent" physical servers, and allow a safe transition of a server from customer A to customer B.

- **Make the BMC more active in auditing the system's TCB.** With secured interfaces and trustworthiness, we could imagine using the BMCs ability to inspect all system state, memory and control even the execution of the operating system, and use that to audit the system at run-time and verify the TCB is kept secure. It's likely that such advanced scenarios will require higher bandwidth than current generation of BMCs have to offer (i.e. faster busses, more compute power).

Summary

In this document, we shared the CSIS's point of view on BMCs. We discussed how BMCs has historically been a security weakness, starting from solving a management problem, but introducing several security risks along the way. We discussed the different components of the BMC stack, and how they could be made more secure, and we recognize some existing industry efforts that are already in motion, working to make BMCs more secure.

Last, we finished this off with a vision of how a trustworthy BMC can be more than just better BMCs. This is an opportunity for BMCs to do much more than they were initially designed for, and focus more on helping platforms become more secure.

We would like to encourage all of those working on various layers of the BMC stack to take this document as input into future planning.

Let's build better BMCs!

References

This document lists many references in footnotes, in context of a specific sentence. The following list contains additional high level references that support the broader context:

1. Cloud Providers Industry Summit (CSIS) charter and prior publications - https://www.cloudsecurityindustrysummit.org/
2. Intelligent Platform Management Interface - https://en.wikipedia.org/wiki/Intelligent_Platform_Management_Interface
3. OpenBMC interfaces - https://github.com/openbmc/docs/blob/master/architecture/interface-overview.md
4. Open Compute Security Specs - https://www.opencompute.org/wiki/Security