The Report Committee for Timothy Michael Lambert
Certifies that this is the approved version of the following report:

Enterprise Platform Systems Management Security Threats and Mitigation Techniques

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Suzanne Barber

Elie Jreij

Enterprise Platform Systems Management Security Threats and Mitigation Techniques

by

Timothy Michael Lambert, B.S.E.E., M.B.A.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2016

Dedication

For Jake Lambert, whose life and passing spurned the author's commitment to enter and dedication to complete this degree program.

Acknowledgements

I would like to express special thanks to my family and employer, Dell Technologies, Inc., for allowing me the time to pursue my educational and professional interests via this superb graduate program. Additionally, I would like to thank my supervisor, Dr. Suzanne Barber, and reader, Elie Jreij, who is a professional mentor in this research area.

Abstract

Enterprise Platform Systems Management Security Threats and Mitigation Techniques

Timothy Michael Lambert, MSE

The University of Texas at Austin, 2016

Supervisor: Suzanne Barber

Developers and technologists of enterprise systems such as servers, storage and networking products must constantly anticipate new cybersecurity threats and evolving security requirements. These requirements are typically sourced from marketing, customer expectations, manufacturing and evolving government standards. Much ongoing major research focus has been on securing the main enterprise system purpose functionality, operating system, network and storage. There appears, however, to be far less research and a growing number of reports of vulnerabilities in the area of enterprise systems management hardware and software subsystems. Many recent examples are within types of subsystems such as baseboard management controllers (BMCs), which are intricate embedded subsystems, independent of the host server system functionality. A BMC is typically comprised of a specialized system-on-a-chip, RAM, non-volatile storage, and sensors, and runs an embedded LINUX Operating System. The BMC's primary roles are always increasing in scope including managing system inventory, system operational health, thermal and power control, event logging, remote console access, provisioning, performance monitoring, software updates and failure

prediction and remediation. To compromise or create a denial of service of such subsystems has an increasing impact on equipment manufacturers and large and small enterprises.

This report's primary objective is to research real-world and theoretical hardware and software cyber-attack vectors on enterprise product platforms, inclusive of BMCs, BIOS and other embedded systems within such products. For each presented attack vector, best practices and suggestions for effective avoidance and mitigation are explored. Domains of particular interest are physical access security, hardware manipulation and secure boot protections against software image manipulation, BIOS recovery and secure field debug techniques.

Table of Contents

List of Tables

List of Figures

Glossary

BIOS	Basic Input / Output System
BMC	Baseboard Management Controller
CPLD	Complex Programmable Logic Device
DMA	Direct Memory Access
eMMC	embedded Multi Media Card
FPGA	Field Programmable Gate Array
FSDAF	Field Service Debug Authorization Facility
GPIO	General Purpose Input Output
iDRAC	Integrated Dell Remote Access Controller (Dell-specific BMC)
HRK	Hidden Root Key
HTTPS	Hypertext Transfer Protocol Secure
IPMI	Intelligent Platform Management Interface
MCU	Microcontroller
SMI	System Management Interrupt
SOC	System on a Chip
SSH	Secure Shell
Uboot	Universal Boot Loader
WS-MAN	Web Services-Management

Chapter One: Introduction

1.1 Motivation

As enterprise systems serve more critical workloads, the enterprise cybersecurity field exhibits a growing level of importance, particularly within local and remote access and systems management. The impacts of enterprise system compromise or denial of service can affect countless end users and astronomical economic impacts. Particular high effect areas are to government and private entities responsible for the reliability, availability, integrity and privacy of content of such systems. Evolving cybersecurity threats require enterprise product developers and technologists to investigate, anticipate and address requirements sourced from marketing, customer expectations, manufacturing and evolving government standards. Although the area of primary focus is enterprise equipment embedded systems, the types of explored attack vectors and suggested mitigations are common to many industries such as client devices, Internet of Things systems and commercial and industrial equipment.

Many recent examples of enterprise system vulnerabilities have been in the area of service processors or baseboard management controllers (BMCs), an intricate embedded subsystem, independent of the host system functionality. Industry experts consider BMCs as high value targets because to compromise or create a denial of service to a BMC can also affect the operation of the host server, inclusive of power down or worse malware infection of the host system through functions such as virtual media redirection.

The typical BMC subsystem is comprised of a system-on-a-chip, RAM, non-volatile storage, and sensors, running an embedded LINUX or real time operating system. The BMC's primary roles are always increasing in scope such as managing system

inventory, operational health, thermal and power control, event logging, remote console access, provisioning, performance monitoring, software updates and fault detection and remediation. Additional examples include other embedded systems such as smart power supplies, storage backplanes, smart fans, network managers and wireless controllers.

1.2 Vision

The vision for this report is to collect through research and experimentation recent real world and hypothetical cyber-attack vectors toward enterprise equipment embedded sub-systems. For each vector suggested techniques based on industry observation, research, government standards or common sense are explored. The goal is to make enterprise equipment manufacturers, customers of such systems and upstream supply chain commodities more aware of and to adopt such requirements and practices. This should significantly enhance the security of such sub-systems.

1.3Scope

The scope of this report entails researching real world and theoretical enterprise product systems management hardware and software cyber-attack vectors, inclusive of BMCs, BIOS and other embedded systems within such products. For each explored attack vector, best practices and suggested methods are presented for mitigating or thwarting each problem. The domains of particular interest are hardware physical access, including port security, hardware manipulation and secure boot protections against embedded systems' code image manipulation, BIOS recovery, and secure field debug techniques. This report explores approximately 30 sub problem areas and 50 mitigating techniques.

1.4 Report Organization

The organization of the remaining chapters is as follows. Chapter 2 is the large bulk of the report exploring various classes of attack vectors with each finding accompanied by one or more suggested techniques for mitigation. Major sections include local physical attacks, compromised authentication attacks and image management. Chapter 3 concludes the report. Finally, chapter 4 provides specific high interest future work area proposals.

1.5 How to Read this Report

The report details 30 cybersecurity problems and 50 best practices and some original solutions to mitigate such attacks. This makes for an average of 1.67 potential solutions for each problem explored with a range of one to seven solutions. Each problem presented will be highlighted and numbered for the reader as follows: (**Problem#)**. Proposed solutions for each problem will also be highlighted and numbered for the reader where each solution will be matched to the respective problem that solution addresses as follows: (**Problem#: Solution#)**.

Chapter Two: Attack Vectors and Mitigation Techniques

According to a 2015 IT survey, 38% more security incidents were detected than in 2014 [1]. Protecting enterprise systems' platform security is commonly a low priority or flatly ignored with respect to physical one-to-one attacks. Many technologists make the decree that if the attacker has physical access to the machine, then there is no security. That is true in some aspects related to highly sophisticated attackers, such as nation states. This paper describes common physical attack methods and ways that users and Original Equipment Manufacturers (OEMs) can design in maximum physical security and obscurity to thwart casual to moderately sophisticated attackers as well.

Next, a common mid-level priority is to protect from remote one-to-one attacks in terms of either system compromise or the instigation of a denial of service. Beyond the obvious perils of compromising that one system, attackers may learn of additional weaknesses or details such as remote administrator login credentials that could lead to the compromise of many more systems. This paper focuses on hardening against some remote attack methods such as in relation to firmware updates and recovery.

Finally, it is a common top priority amongst systems management technologists to protect from remote one-to-many attacks. This type of attack can have large-scale adverse effects on Enterprises, including denial of service, persistent malware and compromise of the host system credentials and data. For example, researchers in one recent study stated: "We conservatively estimate that it would take less than an hour to launch successful parallel attacks against all of the 40,000 ATEN-based Supermicro Intelligent Platform Management Interface devices that we observed listening on public IP addresses" [2].

2.1 Local Access Security

For many businesses, limiting room key or badge reader access and detection via motion and surveillance systems provides sufficient security for physical business critical system access. When such provisions cannot exist, or be trusted, such as in a non-datacenter located server in a retail store, colocation center or even say the back of a military vehicle, local access thwarting methods are critical.

2.1.1 Physical Locks

Most server vendors provide an optional key-locking front bezel or top/side cover. One regular complaint is that the key is typically common between bezels of a given model type, thus only thwarting casual attackers or accidental removal of front hot-pluggable storage drives, button presses, etc. (**Problem1**). Manufacturers may offer re-programmable physical locks, if demand is high enough to warrant such cost (**Problem1: Solution1**). Also, manufacturers could implement a solution where an authenticated remote administrator could physically unlock the system bezel or cover (**Problem1: Solution2**). A quick search of the U.S. Patent Office reveals many applications for remote access systems controlling a programmable electronic lock such as patent number 5,774,058 [3]. None of those found looked to be productized in mainstream enterprise products presumably due to 1) higher cost 2) larger size of a solenoid or similar control apparatus and remote (un)locking system and 3) the general burden of remote network access coordination with the local operator. I believe these impediments could be addressed by electro-mechanical designs which make the remotely controlled physical lock an upsell option off of the base system, such as on a front bezel or cover add-on versus within the base system.

2.1.2 Chassis Intrusion

Most enterprise systems, namely rack and tower servers that are more likely to be located outside of a secure datacenter, support a battery-backed latch and mechanical switch for detecting, logging and alerting administrators of a physical chassis intrusion. When experimenting with a Dell PowerEdge R730 server, which is one of the top selling worldwide systems at the time of this report, it was observed that a chassis cover intrusion while AC power is absent resulted in an event log time-stamp of when AC power was applied and not when the intruder actually removed the cover (**Problem2**). It is a very useful forensic to know when a system cover was removed or replaced versus when it was powered after an intrusion event, such as if there is a supply chain intercept and hardware modification or implant installed. U.S. patent number 6,289,546 teaches a method where a chassis intrusion causes the real-time clock to stop, such that it remains stopped until a software entity acknowledges the intrusion [4] (**Problem2: Solution1**). The only issue here is that time will remain out of date if the system does not have access to a reliable time source such as a network time protocol server or another system within a group (chassis, rack or local area network). It is recommended to use a low power, battery-backed microcontroller or application specific integrated circuit (ASIC) with a real-time clock (RTC), instead of the main chipset RTC, which also can wake and sleep based on GPIO events related to the chassis intrusion assertion and de-assertion (**Problem2: Solution2**). Such a device could log real time stamps of physical access events into non-volatile memory while not losing track of the real world time. At this time, in high volume, such a chip could be easily integrated for an approximately U.S. $.30 per unit adder to the bill of materials. This concept could be extended to accurately capture other useful forensic physical events such as time stamping when AC is lost, which also through R730 system experimentation was not a locally logged event.

2.1.3 Physical and SW Change Logging

Detecting physical and software inventory changes over the life of a product at boot and during runtime is a very useful attack forensic as well for detecting physical and remote attacks (**Problem3**). Fortunately, as of this writing, all tier 1 server vendors have improved on the basic Intelligent Platform Management Interface (IPMI) system Event Log that for many years was limited to 512 system health related events [5]. For example, H-P Enterprise offers an Active System Health Log and Dell offers a Lifecycle Controller Log deep enough to track millions of minute changes over the life of the system [6]. By exporting such logs, an administrator can easily automate the comparison of current and past inventory and user setting changes over time. Additionally, whereas clearing an IPMI system event log to cover an attacker's tracks is very easy with one command, these types of lifecycle logs are intentionally much harder to delete. For example, on Dell servers, a full lifecycle controller wipe is required which is intended when a system is being decommissioned or before being redeployed. Therefore, many other configuration options such as BIOS settings are also purged making a log clearing event quite obvious to administrators. System OEMs and users should constantly seek to enhance the comprehensive event inclusion, protection and use of deep lifecycle logs (**Problem3: Solution1**).

2.1.4 Physical- or Proximity-based External Interface Access

BIOS or firmware enablement/disablement or limited use of physical port access is a common way to protect system operation from physical attacks. As an example, per the Dell PowerEdge latest generation BIOS Setup Guide [7], an administrator can enable/disable the following types of ports:

- Experimentation with a Dell PowerEdge R730 showed that disabling the physical power button results in the server not able to be powered off but can be powered on via the button (**Problem4**). The assumed premise is that interrupting a running enterprise system is bad however booting a powered down system is not an issue due to other assumed protections such as BIOS password or OS login authentication. That may not also be true as a compromised, powered down server, when powered on, could compromise the storage, network or other systems. To address this concern, future systems could offer an additional power button disable option to not be auto-enabled when the server is powered down without preceding authorization via the BMC (**Problem4: Solution1**). This could be performed through remote network interface or local interfaces such as direct connect managed USB, branded "iDRAC Direct", serial or wireless interfaces, if present.
- The non-mask-able interrupt (NMI) button is a common debug feature that can crash the server operating system or interrupt critical processes such as a BIOS update. One area of concern observed on the Dell R730 is that when the NMI button is disabled, a remotely authenticated user can still invoke the NMI via the iDRAC command line or GUI (**Problem5**). Thus, a compromised BMC could crash the host server. If this is a sufficient area of concern, then an additional BIOS option could be added that differentiates disablement of the local and remote NMI buttons (**Problem5: Solution1**). Also during the firmware update procedure, BIOS could reconfigure the NMI pin to disable the interrupt capability.
- External serial ports and managed network ports typically offer static enable or disable options. One suggestion may be to offer an option for authenticated, dynamic, run-time enable/disable of such ports. For example, via the Dell iDRAC, someone

can further disable additional ports and interactions such as whether the LCD is enabled, read only or if it can be used to change configurations and operating states.

- Wireless interfaces such as the Dell Near Field Communication-based QuickSync [8] are becoming more prevalent in enterprise equipment for easing at the box interactions. Various enterprise segments, such as government intelligence agencies, prohibit any wireless communication within datacenters (**Problem6**). With the increasing importance of datacenter system administrator efficiency, it seems as though other wireless or mobile interaction methods may become available in the future. Such interfaces can be disabled within the BMC interfaces, which means they could be dynamically re-enabled. No matter what security guarantees are made by a server vendor, if standard or optional wireless communication features are present, they must be able to be permanently disabled and use strong authentication and encryption best practices in all communications (**Problem6: Solution1**).
- USB ports are an interesting attack vector because of the many use cases that desire more than standard practice basic boot time enablement or disablement of all groups of ports. Some administrators do not trust software disablement of USB ports and physically plug them with hot glue. Where software disablement is trusted, many systems offer BIOS boot menu options to disable all ports or just the rear ports, so that a datacenter technician can use a "crash cart" for front, local interaction. A primary usability issue observed through experimentation is that USB port enablement can only be performed at server boot time via the BIOS setup menu (**Problem7**). This is primarily due to added cost and space of BMC controlled switch logic preventing server host USB controller connectivity to external ports. Vendors of USB host controllers, systems-on-a-chip (SOCs) and CPUs/chipsets should provide an ability via a sideband interface or logic that sits between the host controller and the

user port connector to dynamically control individual USB port enablement at run-time (**Problem7: Solution1**). Thus an authenticated BMC session could enable or disable each port on the fly.

- One solution that exists is the host operating system instituting a filter driver that queries and only allows certain classes of devices such as Human Interface Devices to be exposed to the general operating system. However, this solution is proprietary and thus not available to all operating environments including LINUX [9]. A host OS and driver agnostic solution is highly desired that includes far more powerful pass/fail criteria than a USB device class (**Problem8**). With BMCs being powerful SOCs with a secure root of trust, the BMC could easily play a role in enterprise host USB port device screening (**Problem8: Solution1**). Advanced policies could be set in the BMC to be applied at runtime on existing and newly attached devices. All modern BMCs have at least one USB host controller which could take over external ports on the system when coupled with multiplexer type logic and device attach/detach notification methods. This would allow the BMC to verify, before attachment to the host system, that each attached device meets specific criteria. Some examples include 1) white- or black-listing certain USB device classes such as only allowing a keyboard or mouse and/or excluding mass storage devices, 2) allowing a certain USB mass storage device brands (product ID = SanDisk) or capacities such as 16GB only, or 3) retrieving and verifying from a mass storage device a particular file such as a digital signature, authenticating the device and then allowing it to be attached to the host.

  This is a powerful solution but comes at the cost and space of intervening logic to detect attach/detach events, etc. for possibly many USB ports on a system. A far superior solution involves influencing chipset and USB host controller vendors to add

sideband access for a BMC or an internal configurable policy enforcement management engine. This engine would perform these types of operations directly through each USB host controller hardware, thereby eliminating the need for external logic.

2.2 Internal Physical Attacks

Many leading security experts consistently take the stance that if an attacker has physical access to the electronics or inside of a server, then achieving true security is not possible. This section attempts to investigate and thwart some of those methods used by casual to moderately advanced attackers. Physical attacks create the clear risk of one physical attacker gaining persistent access to a machine that goes unnoticed, such as by intercepting and implanting something within the supply chain or shipment of the system. Such attacks also can enable learning the underlying implementation of an embedded system, such as for gaining root access for launching further remote attacks to many like systems.

2.2.1 Internal Port Access

Servers contain user settable switches or jumpers such as clearing non-volatile RAM or disabling a BIOS password. Manipulation of those have clear effects and user detection such as observed boot messages when testing a Dell R730. For example, the R730 has a local video port disable feature such that local users with a monitor cannot see the video that a remote user is viewing and manipulating. However, the system information label states that the BIOS password enable jumper also re-enables the local video (**Problem9**). This is presumably for cases where an admin loses remote access, such as lost network connection or forgets the remote login and need to see local video in order to reset the remote configuration or login. An enhancement would be to use a BMC

video overlay to require local user authentication before enabling the local host video (**Problem9: Solution1**). The datasheets of mainstream BMCs, such as the Renesas SH7758, describe this capability. Given an input method such as a managed USB port accepting keyboard inputs, the BMC could offer an interactive menu that demands physical access and knowledge such as an administrator password. A solution like this could be used for various security and convenience enhancements. An example may be to externally command operations, such as BIOS NVRAM clear via a local keyboard and monitor versus a multi-step process involving physical dismantling of the system.

Almost all general purpose and application specific integrated circuits (ASICs) have some sort of debug port, the most common being serial ports, Inter-Integrated Circuit (I2C), JTAG (the IEEE 1149.1 standard) or Serial Peripheral Interface (SPI). There is much research on authentication and encryption methods for use of these types of ports but they are generally not in practice amongst most chips. Hardware designers most often just depopulate the relevant connectors in the production printed wire assembly (PWA) bill of materials. There were some recently highly publicized attacks on such ports. For example, the US National Security Agency website published an example implant placed in an intercepted server motherboard that required soldering the removed JTAG connector of the main server CPU (**Problem10**). The same could apply for any such debug port. The implant was an inexpensive, off-the-shelf microcontroller with firmware to customize the exploit as a persistent software application and time delay for launching an attack [10]. New systems should consider adding BMC authenticated enablement of special circuitry that normally blocks electrical connectivity between a target chip such as a CPU or other chip and the associated debug connectors, such as a UART, I2C, or JTAG (**Problem10: Solution1**). Although considered obscurity, this can easily be implemented in hardware, firmware or programmable logic, making it far more

difficult to circumvent than populating a clearly marked standard connector. Techniques can also help obfuscate port enablement, such as embedding signal traces which require X-ray and drilling, and populating false logic that defeats the bus, blowing production fuses, etc.

An additional type of NSA published attack was code named IRONCHEF [10] (**Problem11**). It "provides access persistence to target systems by exploiting the motherboard BIOS and utilizing System Management Mode (SMM) to communicate with a hardware implant that provides two-way RF communication". A similar method could help thwart problem where the BIOS must authenticate with the BMC before the BMC enables circuitry that normally blocks signals as close to the source (chipset) as possible (**Problem11: Solution1**). Since this type of exploit is for attacking the server host during OS runtime, it would be ideal for the CPU/chipset or other IC vendors to enable a lock pin configuration until reset type method (**Problem11: Solution2**). When use of the attached device(s) is complete during boot, the BIOS could request the hardware to lock the pin or bus functionality until a reset occurs. This would render a runtime implant useless. If adopted by IC vendors, this method would require no bill of materials cost adder or added board space.

Nowadays, it is possible for an inexpensive and very small microcontroller (MCU) to be hardware connected to a debug port such as a serial port to listen for and inject certain strings. Persistently implanted MCUs pose a threat if interaction is possible such as stopping that target's boot at a pre-boot prompt or if a root prompt is available via a debug port (**Problem12**). Either pre-set interactions could occur or a bridge to say RF could be part of the implant allowing ongoing remote interaction. Production level firmware should prevent access to a root prompt or limit the capabilities of pre-boot prompt, such as the universal boot loader (uboot) (**Problem12: Solution1**). Firmware

should at least notify another secure agent such as the host server BIOS if access is attempted as detected on for example a receive UART signal.

2.2.2 Silkscreen

Across many industries, printed circuit board designers have a common practice of clearly marking component and connector silkscreen names to meaningful terms. Examples include J_CPU_JTAG or J_BMC_UART for a debug connector instead of using a randomly named reference designator such as J123. Figure 1 shows a couple examples from a Dell server where key UART and JTAG connections are very well labeled for casual attackers.

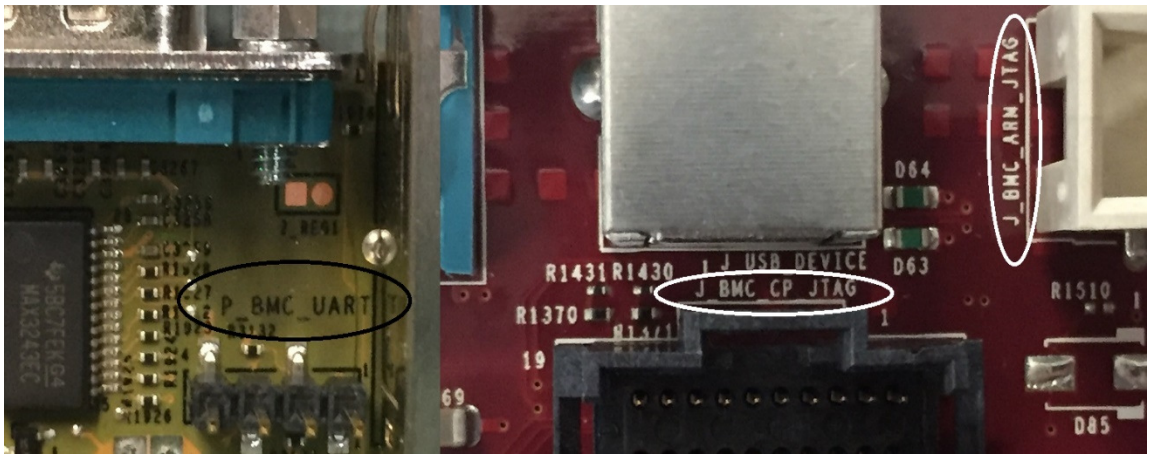

Figure 1 Examples of Non-obfuscated Silkscreen at Critical Debug Ports

The practice of clearly silkscreen marking the purpose of such components eases electronics validation, debug and manufacturing programming. The practice of well-marked silkscreens on critical circuits creates a real threat to even casual attackers (**Problem13**). In one such recent relevant example, a site called "The Ignorant Hack" posted an article about easy hacking of the Dell iDRAC7 [11]. The author noticed the clearly marked "DRAC UART" depopulated four pin connector, which hobbyists know

is almost always receive, transmit, power and ground signals. The attacker then installed into the empty holes an inexpensive, widely available, off-the-shelf UART-to-USB converter and was able to access the service processor debug serial port and discern various details of the underlying hardware and embedded software details. Manufacturer internal groups needing debug connector access possess the design files such as schematics and board layout files with a minor lookup burden versus designing into production a clear path for even casual attackers. Therefore, enterprise equipment vendors should institute a process to obfuscate all critical component silkscreen names either in early development or at least before production hardware release (**Problem13: Solution1**). When surveying a mainstream, tier one server, Table 1 shows some example finds and suggested improvements.

Current Silkscreen	**Suggested Obfuscation**
U_PRIM_SPI_BIOS	U# //BIOS ROM usually contains critical platform data
U_UBOOT	U# //ROM containing bootloader, configuration and log data
P_BMC_UART	P# //The BMC's debug serial port input/output are possible
J_BMC_ARM_JTAG	J# //BMC main JTAG
J_BMC_CP_JTAG	J# //Co-processor JTAG
PEMMC_DBG	P# //A port for accessing an eMMC Flash device that holds the BMC's operating system and other functions. A USB card reader attached to this port could observe or manipulate the Flash contents.
J_CPLD_JTAG	J# // A middle man attack could overwrite a critical programmable logic device.
SW_IDRAC_PORST	SW# // An attacker could inject a BMC power on reset which by experimentation shows to crash the host.
J_MFG_MODE	TP# //Shorting such jumper holes with a paper clip or tweezer may put a subsystem in a non-customer mode. Hardware designs could randomly number or require something more difficult than a simple ground, such as a specific pulse stream or loadable license to enable.

Table 1:Examples of a mainstream server's non-obfuscated silkscreen labels on critical circuits and improvement suggestions for manufacturers.

2.3 Remote Attacks

As mentioned in the introduction, remote attack vectors can be far more severe than physical attacks due to the one-to-many, parallel nature of such attacks and due to possibly less detection forensics. This section discusses some of these attack vectors and possible enhancements for manufacturers or end users.

2.3.1 Default Passwords

For many years, enterprise server systems would ship with BMCs and other service processors with common default root passwords. Dell BMCs supported login of "root" and password of "calvin". Other vendors published similar defaults such as login "admin" and password "admin". This poses a problem when simple to obtain tools like the one referenced here [12] are used to scan networks such as the open internet for BMC interface protocol responses, such as HTTPS and SSH, and attempts to login with the default credentials (**Problem14**). According to one research paper, one internet scan revealed greater than 105,000 servers with their BMCs with default credentials connected to the public internet and thus vulnerable [13]. Once a remote user has access to the BMC, even without root level administrative rights, many possible attacks on the host server are possible, even without host server credentials. For example, one could remotely attach USB-based virtual media which could install malware or rootkits to the host.

Besides using good networking techniques such as firewalls and VLANs and not connecting subsystems such as BMCs directly to the public internet, manufacturers should offer an ability to permanently alter the default password (**Problem14: Solution1**). This helps such that a reset to factory defaults operation does not inherently

alter root access credentials to something widely known and common. Some vendors such as H-P Enterprise look to already support such capability as described in [14].

Manufacturers should also consider making the default passwords unique to each unit of each model shipped such as based on the unmodifiable service tag (**Problem14: Solution2**). From some documentation such as a Dell OEM called ExtraHop's documentation [15], top tier server vendors already offer this capability, presumably in response to thwarting the type of issue described here.

Anytime a default password is in place on a system, it should be made clear to the user such as via BIOS boot messages, web GUIs, notifications and command line interfaces upon logging in (**Problem14: Solution3**). Articles such as a Dell Techcenter Blog article related to iDRAC7 [16] imply that such warnings are becoming more prevalent in new generations of products and/or firmware releases. Some security technologists that I interviewed about this topic have strong positions to absolutely force the user to alter the default password before the system is fully usable versus just a warning.

2.3.2 Protocol Attacks

In the early 2000's, IPMI was the only widely used protocol for managing server BMCs. Since then various other protocols have become prevalent, including web-based HTTPS, secure shell (SSH), Web Services-Management (WS-MAN) and the emerging Redfish standard [17] developed and managed by the Desktop Management Task Force (DMTF). Insecure methods such as Telnet and HTTP have been almost entirely deprecated within the enterprise systems industry, which is a very good thing for security. Even standard secure network protocols suffer from evolving and more creative attack

vectors (**Problem15**). For example, the Heartbleed Bug [18], was a critical security bug for all applications that used the affected standard protocols such SSH.

Manufacturers should be very wary about producing their own communication methods versus leveraging the widespread diligence and patching on standard protocols (**Problem15: Solution1**). The customization benefits seem to always get questioned when any security vulnerability is discovered.

With constantly evolving attacks on standard protocols, it is extremely important that manufacturers stay vigilant with releasing timely field programmable updates, such as BIOS and BMC firmware (**Problem15: Solution12**. Customers should also frequently monitor the change lists on releases, apply any updates/patches, even if say a server is only on an internal network, and finally demand of enterprise manufacturers to patch known vulnerabilities as soon as possible. A couple examples for firmware developers are the always keeping up to date with kernel security patches and utilize the most recent services such as the later Apache web server version.

Even within standard systems management methods such as IPMI, manufacturers generally advise customers to not put IPMI enabled systems on the open internet but instead on a firewall protected VLAN (**Problem15: Solution3**). A 2013 Arstechnica article [19] referenced researchers estimating very easy wide-spread parallel attacks on any thousands of IPMI devices on the public IP addresses.

Despite security sensitivities, there continues to be expansion of wireless communication uses in datacenters for mobile device to server communication and for functions such as asset location (**Problem16**). For example, Dell's 13th Generation servers offer Quick Sync which is an optional module that offers Near Field Communication (NFC) between the BMC and mobile devices. It is conceivable as evidenced in simple google searches that other standard wireless protocols could/will be

used in some datacenters. As stated in the physical port section, any wireless communication method must be an optional upgrade and not a required capability as many segments such as government may flatly outlaw wireless use (**Problem16: Solution1**). Manufacturers who produce wireless solutions must keep up with the latest wireless attack exploits and be quick to implement and roll out software/firmware fixes (**Problem16: Solution2**). Additionally, best practices should be adopted for authentication and pairing of the server and diverse types of client devices.

2.3.2 Credential Vault

Establishing secure protocol sessions such as remote SSH, HTTPS, etc. requires key exchanges between the initiator and the BMC. BMCs must inherently store cryptographic private keys in non-volatile memory which must remain secret even if ROM access is achieved, such as by physically de-soldering and de-capping the ROM device package (**Problem17**). Dell's iDRAC Credential Vault and Hidden Root Key (HRK) are good examples of embedded system capabilities that work in concert to 1) establish a secure root-of-trust in the boot sequence, 2) provide assurance that code running on the BMC is securely signed and 3) ensure that directly read ROM contents are encrypted [20]. The hidden root key consists of each iDRAC containing a unique 256-bit binary value burned into the silicon that is hidden from software and available as an input key to internal cryptographic acceleration engines. The datasheet of a shipping, mainstream BMC, the Renesas SH7758, was reviewed. If the boot sequence is determined to not be a root of trust or if a JTAG connection is detected, then the Hidden Root Key is disabled which prevents the credential vault from being decrypted.

Enterprise system manufacturers should use BMCs that provide the facilities for hidden root keys, internal ROM code that can support a secure boot chain of trust and

facilities to detect common attack types and disable the hidden root key (**Problem17: Solution1**). Security processor targeted SOCs have had this capability for a few years. This capability definitely should propagate to many other chips and applications, such as IOT targeted microcontrollers. This may seem like an obvious requirement going forward, but as described in a later section, this capability is currently only in one of three main shipping BMCs and in very few other SOCs and MCUs.

2.4 Design for Manufacturing and Production Debug

Balancing platform security with enterprise manufacturing and debug needs during manufacturing, failure analysis of field returns and engineering sustaining efforts with production level hardware, firmware, BIOS and other programmable images is a significant development challenge. The impact of an enterprise device being compromised can be much worse than on a client device, such as a notebook or desktop computer. If a process or consistent portfolio guidance adopts a more inward focused solution, then internal groups can be effective and streamlined when access is needed at the detriment of production security exposures. Adoption of a security-focused conservative approach creates more difficult and time consuming situations such as requiring physical hardware manipulation and/or modification of loaded programmable images to debug hardware or firmware issues or re-deploy field returned hardware. Another big concern is that modifying the loaded programmable devices, such as by creating and updating signed, non-production firmware images, loses the current failure state of the suspect system or can easily make the issue no longer repeatable. Both of these are problematic in emergency customer escalation type scenarios because of the added debug and setup time. A hardware manipulation based method to invoke a debug mode may not even be possible within the full system chassis unless very much focused

on during the early design phases. This section focuses on enabling enterprise system developers to effectively and efficiently perform the necessary manufacturing functions and issue resolution tasks, while balancing sufficient production system security tenets.

2.4.1 Field Service Debug Authorization

During enterprise systems development, firmware build flags are typically set that enable full debug capabilities. Where security issues are anticipated, many debug capabilities are turned off in production firmware so that attackers cannot for example 1) access debug port inputs or outputs, 2) alter the boot path, such as stopping at a boot loader prompt to manipulate states or implant malware or 3) gain root prompt access to the embedded operating system as a whole.

Production support and engineering teams typically need to access advanced debug capabilities with production hardware and firmware while not compromising mass volume security (**Problem18**). For example, a difficult to debug issue in a customer's large datacenter may require access to special logs that are not normally customer extractable. Furthermore, manufacturers enabling undocumented mechanisms to achieve such backdoor type access violates the major tenet that obscurity is not security.

Dell has addressed this issue in a unique way that others may learn from with their iDRAC Field Service Debug Authorization Facility (FSDAF) [20] (**Problem18: Solution1**). This facility utilizes a mechanism where both the end customer and manufacturer authorize specific debug capabilities, including up to root access, for specific durations on a particular system. A signed certificate is then uploaded into the iDRAC enabling the agreed upon features. Thus, the mutually agreed upon access level is authorized and dynamically achieved for the agreed duration without having to modify

the firmware or even reboot. Through experimentation on a Dell R730, I observed that FSDAF is enabled via an uploaded certificate into a fully operational and booted iDRAC.

FSDAF is great for debugging application level issues. However, various common debug needs are not satisfied with FSDAF such as an embedded OS that crashes or fails to boot or other low level issues (**Problem19**). Manufacturers should enable an enhanced solution where a cryptographically sound method for debug capability authorization can persist through a reboot or ideally be accomplished in the early boot stage, such as the boot block (**Problem19: Solution1**). This would give field access, for example, to a Universal Boot Loader (uboot) prompt to help debug low level types of issues or recover a non-authentic or corrupt embedded subsystem OS image without needing to update to a non-production, signed firmware image.

2.4.2 Manufacturing Mode

The manufacturing environment of complex enterprise printed circuit boards requires special privileges that customers should never have, such as for setting persistent networking MAC addresses and blowing one time programmable fuses such as cryptographic keys. Note that it is typical to fuse several public keys in case, for example, a corresponding private key is compromised, then the system can still be securely updated with a new verified image utilizing an alternate key.

When production support for embedded systems' elevated debug privileges are required, then it is wise to provide an immutable indication to another secure entity, such as BIOS (**Problem20**). That entity could then take appropriate actions like notifying the end user and attempting to remedy the situation once the access window expires. Examples of useful elevated privileges include 1) output debug serial port spew of the boot block, uboot execution, LINUX OS boot, kernel panic messages and BIOS debug

boot messages and 2) interaction with a uboot prompt. Embedded OS root access should not be supported through obscurity, not matter what. Thus, root access should be excluded from production images and manufacturing planning.

The factory test and service support staff that I interviewed at Dell stated a clear dissent of supporting multiple firmware images, especially a non-production, elevated privilege version. The concern centers on the opportunity for a human mistake that lets non-production code get out of the manufacturer's control. Further, firmware capabilities such as limited option menus or limited command line interfaces are possible for example in uboot. This however creates extra development and validation effort and are limited to the hard coded capabilities provided. Therefore, manufacturing enhancements would require firmware changes.

As described earlier, an obscure physical manipulation to invoke manufacturing mode should be avoided such as grounding a circuit board test point (**Problem20: Solution1**). Early boot code, such as the mask ROM or hardware logic, should set a one-time per boot immutable bit or flag to another secure agent, such as BIOS. This bit or flag should ideally reside inside the BMC or embedded controller chip, to avoid external bus manipulation or snooping, but optionally also in external logic such as discrete gates or a CPLD or FPGA. This bit or flag should be clearable only by a power-on-reset. This is because a power on reset guarantees all RAM is re-initialized, whereas a core reset does not. If this bit could be cleared or the full RAM not re-initialized, then an attacker for example could stop at a uboot prompt and alter the boot path or install malware into memory before continuing the secure boot path. This mechanism allows the secure agent to detect this operating state, notify the user of the possibly insecure operating state and attempt to heal the situation by issuing a power on reset to the reporting entity.

2.5 Image Boot, Update and Protection Management

One of the most critical security areas of embedded systems in general, including enterprise product subsystems such as BMCs, relates to image verification during the updating and booting processes. This section explores issues and best practices related to image verification, achieving a secure boot path, ROM protection and embedded data handing during system decommissioning and re-provisioning. Even though BIOS is not technically part of the systems management subsystem, a platform security treatment would be incomplete with investigating BIOS secure boot issues and the BMC's involvement with BIOS recovery solutions.

2.5.1 Image Verification

It is highly valuable to adopt image signing and image verification as early as possible in the software development cycle for practice with the build, signing and verification procedures and to get the implementations for setting and revoking the proper keys in place.

Image verification is important to be performed before being committed to non-volatile storage (**Problem21**). Even though it is mostly standard practice to do this with BMC firmware and BIOS, it is far from standard in other embedded systems. Released images should always contain a public key and be signed by the manufacturers private key through a tightly controller process. During update, the receiving firmware entity verifies authenticity of the image utilizing the public key. Dell's process for image signing as described in [20] looks to conform to industry best practices. Manufacturers should strive to have 100% coverage of image verification as the image arrives at the destination or at least within the system (**Problem21: Solution1**). This should include not just BMCs but also for example controllers with firmware in backplanes, power

supplies, voltage regulators, storage controllers, control panels, wireless controllers, etc. Chip vendors more likely need pressure through request for information (RFI) and request for quote (RFQ) efforts to add the necessary hooks for endpoint image verification. Luckily such security capabilities are generally being requested of these types of chips, ROMs and firmware as a basic need for many embedded applications including Internet of Things (IOT) applications.

The entity being updated, whether it be a small microcontroller or a higher end SOC, must verify the image upon receipt before committing to nonvolatile memory. This is not always a possibility. For example, most microcontrollers possess a firmware binary image larger than the amount of RAM available. Thus staging the entire image for an authenticity check before writing to nonvolatile storage is not possible (**Problem22**). A common challenge for hardware and system designers is right sizing the RAM and ROM needs of embedded systems and specifically microcontrollers and SOCs such as those used in enterprise systems like backplane controllers, power supplies, smart fan controllers, etc. For example, it is very common to not have equal RAM and ROM space as RAM is much costlier than FLASH in integrated circuit space. A 1:2 ratio of RAM to ROM is very common in the current MCU industry. Additionally, system designers usually cannot spend more money on a chip(s) with twice the ROM space than needed under normal operation. It is not a good solution to verify the image by the receiving device on one transfer, then request another transfer that gets committed to the ROM, because the source updating entity could replace the image on the second transfer.

For controller/subsystems with enough non-volatile storage to store multiple image copies, firmware could commit the image to the primary location (**Problem22: Solution1**). If verification fails, then the embedded firmware could flag the failure but

then autonomously copy the previously verified backup image to heal the primary image location with the original image.

Hardware applications that cannot stage the entire image before commitment to the non-volatile storage could follow the original flow in figure 2 (**Problem22: Solution2**). After a trust relationship is made between the updater entity and the updated entity (endpoint), the updater could retrieve the current image from the endpoint to heal/revert back to it if the new image fails to verify authenticity and integrity. This allows the endpoint to commit the new image in chunks without fear of being persistently without a bootable image.

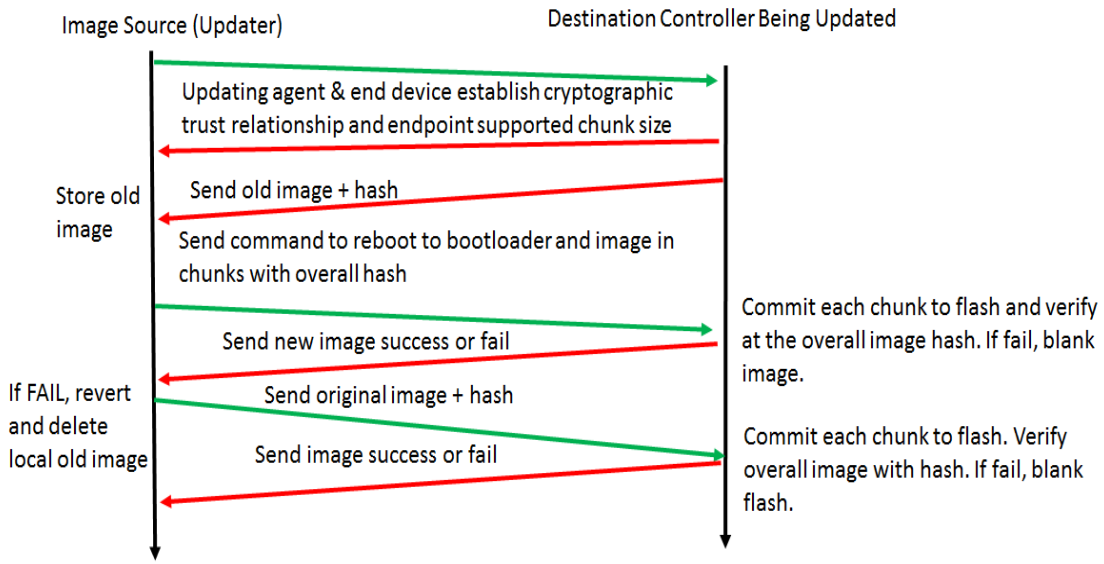

Figure 2 ROM Space Optimized Image Update and Verification Method

2.5.2 Secure Boot Path

The importance was emphasized earlier of a hidden root key, internal security ROM code and secure chain of trust boot sequence in the context of a credential vault. These are also critical to achieving a secure boot path solution. In experimenting with a

mainstream server, a secure boot failure of the BMC image properly invalidated the Credential Vault, protecting the secrets, but one of the firmware images booted anyway. This may help with availability such as of thermal algorithms running in the main operating system. It is not however the most secure behavioral policy such as halt on failure, or try to auto-recover from a previously verified image.

Few BMCs in the industry nor many available microcontrollers support the necessary hardware capabilities to inherently build a chain of trust. Even through BMCs have such a pivotal role in the datacenter, I compared the datasheets of the three shipping BMC vendors' latest chip offerings which comprise an overwhelming majority of enterprise systems in recent years: Aspeed AST2500, Emulex Pilot 3 and the Renesas SH7758. Renesas was the only one with an HRK and internal ROM code on top of which a secure boot chain-of-trust can be established. This means that much more of a vulnerability exists on systems with BMCs without such capabilities. A cursory look showed that many systems utilizing the non-secure boot capable Aspeed BMC were from vendors such as SuperMicro and hyper scale systems such as from Facebook's Open BMC initiative effort [21].

Unlike less critical embedded systems, enterprise subsystem applications' need for high availability often warrants a redundant boot path. This is to account for possible compromise or corrupt images in the boot path. Corruption could occur in cases such as power loss during firmware update. The basic flow for figure 3 includes a primary and secondary boot path where each entity verifies the authenticity, and inherently the integrity, of the subsequent chain before passing off control. There are alternative paths possible but this is one that prohibits altering the boot path mid-stream. That allows for alternate versions of firmware between the two boot paths to facilitate features such as version rollback.

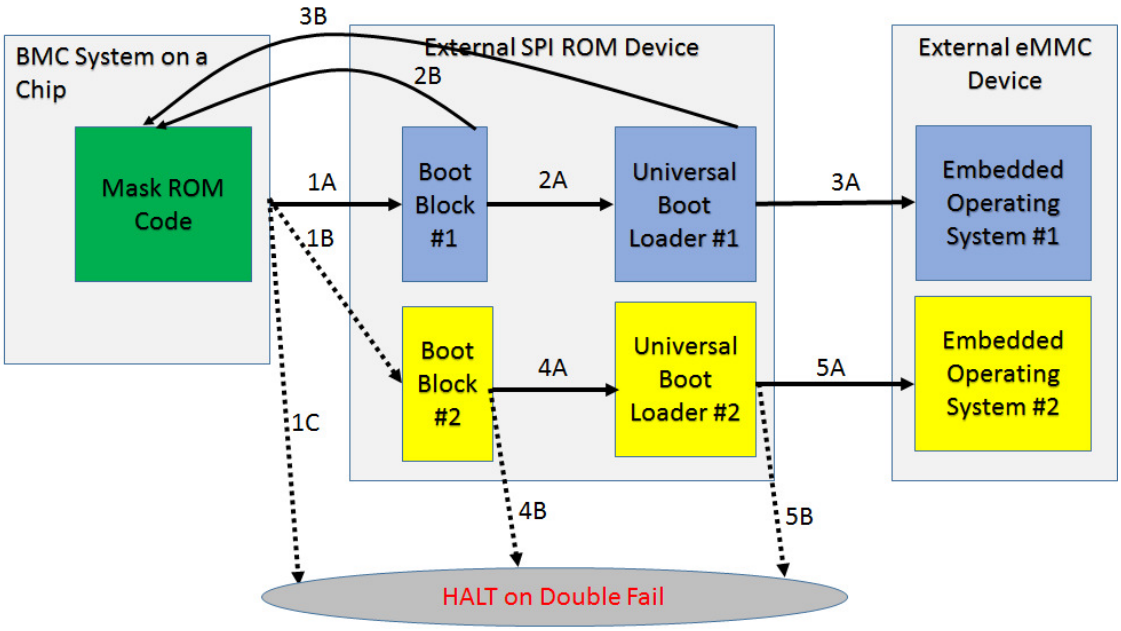

Figure 3 Example Secure Chain of Trust Boot Flow with Redundant Images

Step	Detail
1A	Mask ROM validates the primary Boot Block before handing off control.
1B	If 1A fails, Mask ROM validates the secondary Boot Block before handing off control.
1C	If 1B fails, Mask ROM must halt on fail, as jumping to untrusted code is unacceptable.
2A	Boot Block validates the primary Boot Loader before handing off control.
2B	If 2B fails, Boot Block sets a flag and jumps back to Mask ROM to try the secondary boot path.
3A	Primary Boot Loader validates the Primary embedded OS before handing off control.
3B	If 3A fails, Primary Boot Loader sets a flag and jumps back to Mask ROM to try the secondary boot path.
4A	Secondary Boot Block validates secondary Boot Loader before handing off control.
4B	If 4A fails, Secondary Boot Block must halt on fail, as jumping to untrusted code is unacceptable.
5A	Secondary Boot Loader validates Secondary embedded OS before handing off control.
5B	If 5A fails, Secondary Boot Loader must halt on fail, as jumping to untrusted code is unacceptable.

Table 2:Example Secure Chain of Trust Boot Flow with Redundant or Alternate Images.

One alternative when using a BMC or embedded system which does not provide internal root of trust capabilities, is to utilize external, intermediary chips that do. Such a solution has the hardware root of trust sitting between the main processor and the non-volatile storage such as the boot SPI ROM with a mechanism to reset the main processor if not deemed secure. One such example is described in a recent Microsemi whitepaper [22]**.** These solutions are very costly due to the dedicated silicon and package addition. They are also physically large which hurts dense hardware applications. Finally, they hurt overall reliability of the system by adding another active component with internal firmware that could itself fail or need to be firmware updated.

2.5.3 Embedded Non-Volatile Storage and I/O Protection

During boot and runtime of an embedded system, protection of the non-volatile memory locations where the boot image(s) and other static, non-volatile data reside is a critical security need. In addition, in most modern chip and software architectures, code running on a particular domain, such as a BMC CPU or host server main CPU, has unrestricted access to the underlying hardware. In this section, we explore specifics in these areas for common device types and architectures, as well as explore the "app store" concept's issues unique to embedded systems.

2.5.3.1 SPI ROM Data Protection Methods

In a large portion of embedded systems, all of the execution code is in an internal or external ROM(s), which in most current applications, utilize ubiquitous serial peripheral interface (SPI) ROMs with high endurance NOR FLASH technology. In smaller embedded systems, the entire code base resides in the SPI ROM. In more complex BMC type applications, such as those running embedded LINUX, the SPI ROM is used to hold the boot block(s) and boot loader(s) and other user data such as logs. The

large operating system image(s) and other large partitions, which often approach or exceed approximately 128MB, reside in managed NAND FLASH devices like eMMC.

Redundant code images are often utilized for high availability in case of software corruption, bad FLASH cells in the hardware or malicious modification of the image(s) (**Problem23)**. However, that method cannot be a critical subsystem's only defense.

System designers should definitely ensure the SPI ROM contents are write protected by the boo tloader before handing off control to the operating system and that the write protection cannot be circumvented until a reset occurs (**Problem23: Solution1**). During firmware updates, uboot should be updated only from uboot. This guarantees preventing a compromised OS or accidental firmware bug from being able to write to the critical boot image locations.

The ideal SPI ROM write protection method is to include fixed logic in between the internal chip CPU bus and a generic SPI controller (**Problem23: Solution2**). An example is logic that allows firmware to populate a lock-until-reset whitelist (preferred) or blacklist (if there is a sufficient number of op codes that can be blocked across vendors) of SPI operation codes over specific address spaces of interest. For example, if a 4MB SPI ROM is used where the boot loader(s) reside in the first 1 MB, then such a method could allow only intended write, program and erase commands to reach the ROM if there is an address range match. This capability exists in some advanced chips, but is very far from being standardized. Requests for quotes for new enterprise applications should explicitly include this to get more vendors to add such capabilities.

When the chosen BMC, SOC or microcontroller does not provide the logic highlighted above, then SPI ROMs themselves support write protections on a sector basis (**Problem23: Solution3**). One major challenge for hardware and firmware developers is that the SPI ROM industry has no standardization between vendors on methods to enable

such protections. In a review of five leading vendor datasheets for protection schemes, each one had variations. For high volume production of long ship life enterprise products, it is common practice to qualify at least three different vendors for cost and continuity of supply. An example difference may be that one vendor part supports a full device erase op code that would need to be blocked specific to that part. This appreciably complicates the firmware and requires the developer to read through literally thousands of pages of datasheets and compile comparison charts for firmware to code around. This is needed to ensure security protections are covered and production affecting mistakes are not made such as accidentally permanently write locking a sector versus only until the next power cycle. Another challenge is that SPI ROMs generally do not have reset inputs to remove the write protections. This implies the system orchestration or hardware such as localized SPI ROM power cycle coincident with a processor reset is needed at extra cost, logic and board space.

2.5.3.2 Managed NAND Flash Data Protection Methods

Many embedded systems such as BMCs also need large NAND flash devices, which most of the time are soldered to an expensive motherboard. System designers need to have utmost focus on the protection of non-replaceable embedded flash devices from accidental or malicious overwrites, unsecure field firmware updates (FFUs) and partial data extraction at any point (**Problem24**). At the time of this report, eMMC is the leading standard for embedded NAND flash devices, where the smallest purchasable capacity is 4GB. These types of devices have internal controllers between the bus interface and the actual NAND flash cells for performing security functions, wear leveling, error correction, etc. In BMC type applications, the data stored in eMMC includes multiple

copies of the embedded LINUX operating systems (>100MB), system diagnostics, lifecycle logs, OS device drivers and pre-boot utilities.

Firmware developers should use the well-established LINUX methods for converting the necessary file systems to read only so that a compromised OS would still block writes to critical areas (**Problem24: Solution1**).

The latest versions of the eMMC specification added permanent and power-on eMMC sector based write protections [23] (**Problem24: Solution2**). Permanent write protection should only be used when absolutely necessary due to possible field issues if an update were needed. Write protect until power cycle is a very useful capability for firmware updates.

In addition, the eMMC specification added a hardware reset pin. This can be pulsed by system logic in conjunction with the BMC reset to intentionally remove power-on write protections when needed (**Problem24: Solution3**).

System designers must guarantee that embedded NAND FLASH devices are not excessively worn out (**Problem24: Solution4**). A wear model should be created to understand the typical and worst case software use cases and how they affect writes to the embedded flash device. Developers should beware of small and large size write patterns and the file systems used as they can significantly affect write amplification. The firmware device driver should instrument counters on reads and writes for validation of real operations to the expectations in the model. Future methods and algorithms could be implemented where run-time agents observe the statistics running on a BMC in real time and then adjust the usages or warn users of excessive use beyond design expectations. This helps thwart bugs and user unintended or attacker-initiated FLASH write abuse. Finally, enterprise system(s) should be validated against an accurate intended wear model

to ensure excursions do not get into released code. This last part ideally is performed on a regular, automated build verification test scenario.

Designers should increase part reliability by running eMMC devices in pseudo-SLC mode (**Problem24: Solution5**). This provides more resilience to part write abuse and means that a designer can use less expensive but less reliable multi-level cell (MLC) devices in pseudo-single-level cell mode (pSLC). In pSLC, the analog to digital converter can read a wider voltage to determine the intended value, thereby increasing the write endurance significantly [24]. In comparing three leading eMMC device datasheets, pseudo-SLC mode reduces overall capacity by approximately 60% which often is not an issue at the embedded system storage needs pale in comparison to the smallest procurable managed NCNAD devices. pSLC also improves the program/erase cycle guaranteed limits from about 3,000 to 30,000-50,000, or more than 10X more reliability.

With managed NAND, wear leveling and bad block management abstracts physical addresses from logical addresses and thus may leave partial files/data intact in physical locations following logical address erases (**Problem24: Solution6**). During system wipe type activities, firmware should take advantage of the eMMC secure erase command, where "data in the specified memory addresses must be purged from the physical memory array" [23]. An additional assistive feature is called secure trim which is effectively secure erase performed at the sector level.

The latest eMMC revision also supports Field Firmware Update, where the device manufacturer may want to, for example, improve their wear level algorithm after shipment or fix a bug (**Problem24: Solution7**). Since these types of devices were not updateable prior to this eMMC specification revision, this imposes a new security concern. Like mentioned in an earlier section, system designers should ensure that the updating entity such as the BMC verifies the signature of the eMMC field firmware

update image before applying it to the target device that cannot perform authentication operations on its own.

2.5.3.3 The "App Store" Concept in Embedded Systems

Most embedded systems including enterprise BMCs are generally considered a closed system. This means that a user cannot normally modify the operating environment. However, there are markets such as amongst original equipment manufacturers (OEMs), where generic enterprise systems are rebranded and settings customized. There are recurring proposals for a BMC "app store". A BMC "app store" where aftermarket code would run on a BMC or other management controller that is not part of the base firmware creates a significant security and system stability concern (**Problem25**). There is also a real risk to the manufacturer, warranty and stability of the BMC and host system when allowing running not thoroughly validated code on the same service processor and embedded operating system that must also perform numerous critical near real-time functions including power and thermal controls.

If an "app store" is a marketing requirement, designers should limit the "apps" to scripts that are equivalent to calls that remote, authenticated users and consoles could execute and thus validated as part of the base system (**Problem25: Solution1**). In an extreme case, where injecting aftermarket code into a BMC domain must be supported, ensure that it is a user space application without kernel space privileges. Also, consider instituting an application monitor in the base firmware, which observes whether new processes misbehave. A couple examples are the app consuming excess CPU cycles or memory, inclusive of slow memory leaks for extended periods. Applying embedded systems concepts such as virtualization or containers may prove valuable.

2.5.3.4 Secure Peripheral and GPIO Access

Many types of bus peripherals in most modern chips are memory-mapped interfaces that are open for access by locally running software. One example is an I2C bus controller that connects to sensors and control functions such as clock chips. Another key example is a GPIO controller, which provides expandability to much control and status instrumentation in a system. A typical server may have >256 GPIOs, where the possible exposure of improper (re)configuration could be catastrophic to system operation or performance. Finally, for maximum flexibility, almost all modern chips offer multiple functions on one pin, whereas the control for changing that functionality at runtime is not typically lockable. As an example, in the most recent shipping enterprise chipsets from Intel [25], GPIOs are not lockable or protected and pin function is changeable from a compromised operating system (**Problem26**).

For static GPIOs and pin functions whose is static at runtime, a granular lock-until-reset control method is needed in silicon that make various register bits write once instead of the typical free read/write capability (**Problem26: Solution1**). When utilizing programmable devices, such as CPLDs and FPGAs, it is common practice to implement this type of method with custom code. However, this capability is far from standard, sufficiently granular, nor pervasive enough in today's commodity microcontrollers, or even more sophisticated BMC systems-on-a-chip or chipsets.

When silicon cannot be changed, then another solution is to rely on another secure entity to monitor for state changes (**Problem26: Solution2**). For example, modern Intel chipsets have a management engine running firmware inside that can also access various registers such as GPIOs and I/O pin functions. Before handing off control to the host operating system, booting BIOS could tell the management engine which I/O and pins must be locked until the next platform reset. After that, if they are observed to be

changed, then the management engine could restore the unexpected modification and alert the BMC for logging or other remediation. This would be an improvement but not ideal due to the effects of momentary blips.

In some cases, local GPIO states need to legitimately be altered during runtime, such as by BIOS while servicing a systems management interrupt (SMI) (**Problem26: Solution3**). Cross domain authentication is ideal for an entity that wants to change its local configuration to confirm permission from another secure domain. Development of a request / grant protocol proving authenticity of the requesting entity would go a long way to hardening this vulnerability. Such a solution would require modifications to the silicon such as hiding the chipset GPIO register space from the host OS. How to ensure that the request is from a trusted versus compromised source entity is the challenge. e.g., Could the key for an encrypted command have been compromised? At a minimum, the BIOS could send a subset list of pins/functions that need to be runtime modified to the management engine at end of boot, thereby at least limiting the grants to those requests. Exploring solutions from other domains should occur to create a foolproof solution.

2.5.4 System Decommissioning or Re-provisioning

Proper information handling toward the end of a system's life or when being re-deployed for alternate functions or refurbished is an opportune time for enterprise system owners to make critical mistakes. There is much research and best practices related to computer hard disk drive secure data wipes, disposal and self-encrypting drives to protect the main server use data. Data sensitive customers also have great interest in the data held in various other embedded system ROMs. Manufacturers are typically required to publish a statement-of-volatility to provide such transparency. For system decommissioning and/or re-deployment activities, system manufacturers must provide customer

satisfactory assurance that all non-volatile storage elements with end user modified data have been thoroughly erased and settings reset back to factory defaults (**Problem27**).

For example, Dell not only publishes statements-of-volatility for all enterprise systems but also provides a "System Erase feature as part of the iDRAC with Lifecycle Controller (LC) embedded systems management solution" [26] (**Problem27: Solution1**). The main capability is called System Wipe allowing granular and user-selectable categories of deletion. Example categories of operations include erasing logs (including the afore mentioned comprehensive lifecycle log), BIOS and BMC configuration data set back to factory defaults, embedded RAID controller cache erase, and managed persistent storage formatting.

2.6 BIOS Secure Boot, Failure Detection and Recovery

Though the majority of this paper revolves around enterprise server management subsystems, it is also important to explore the system BIOS security protections. In recent years, numerous articles discuss ever evolving attack methods on computer BIOS. The protections explored include BIOS secure booting, executing only authentic code, BIOS update, measurement, altered image detection and secure recovery.

2.6.1 UEFI Secure Boot

BIOS secure boot in the enterprise is a must have offering for practically all new products (**Problem28**). Modern enterprise systems base their BIOS on the Unified Extensible Firmware Interface (UEFI) specification [27]. An optional feature of the UEFI specification is UEFI secure boot and is offered as a customer option in many modern server systems, although the implementation specifics varies. Similar to secure boot in

embedded systems, the secure chain of trust starts with a small piece of immutable code running typically on an embedded security processor inside the chipset. Then "each subsequently-executed section of code is verified safe and unmodified before it is executed" [28]. Reference 28 titled "UEFI Secure Boot in Modern Computer Security Solutions" is an excellent overview of this area.

Intel's commercial client and enterprise chipsets accomplish this via a feature called Intel Boot Guard [29]. Controlling logic inside the silicon "verifies a signature contained in the firmware image before executing it, using the hash of the public half of the signing key, which is fused into the system's Platform Controller Hub (PCH) by the system manufacturer." Other complex chip makers offer similar features.

System designers should take full advantage of these offerings when available, as well as enterprise system administrators for ensuring optional security features are enabled, which may not be the factory default mode of operation (**Problem28: Solution1**). When these features are not available, technologists and business requests for quote should demand these in additional applications.

2.6.2 BIOS Recovery

Modern chipset ROMs contain many critical functions needed by the CPUs and chipset beyond just the BIOS image, including soft straps, management engine firmware, network peripheral option ROMs, etc.

When a secure boot failure occurs, the policy is typically to halt-on-fail rendering the system useless (**Problem29**). That is because the provision to have a recovery image to autonomously failover and heal the primary image is complex. It also consumes large amounts of storage space to house an additional redundant primary image. Most enterprise systems have secure subsystems such as BMCs that should seemingly always

play a role in customer notification and user initiated or autonomous policy based BIOS image recovery (**Problem29: Solution1**). Thus, secure failure detection should be indicated through I/O or ideally through a sideband register or command, such as the traditional port 80h codes used for host boot status with granular failure reasoning.

For various reasons promoting best cybersecurity practices, the U.S. National Institute of Standards and Technology (NIST) has published the BIOS Protection Guidelines for Servers [30]. These guidelines specifically place a couple key constraints on a BMC or service processor that has access to the BIOS ROM device or contents for purposes of BIOS update. The first is that the BMC environment may be employed as a Root of Trust for Update (RTU) for the system BIOS if the BMC is guaranteed to be updated and booted via authentic code. Authorization to execute such tasks is also required. These are very important because BMC adding value in BIOS recovery cannot open up new attack vectors. Since an earlier section suggested the BMC be a secure root of trust for itself, the BMC's need to be a (RTU) is not a unique suggestion here.

The second NIST requirement is that "the Service Processor [or BMC] shall not have direct and unrestricted access to system memory on the server outside the control of the host operating system, to prevent the SP from interfering with legitimate update processes." BMC and other embedded systems chip designers and hardware and system designers must ensure that there is no direct host memory access in an unrestricted sense in platforms where direct host interfaces exist such as PCI Express, USB, etc. or sideband accesses such as via I2C (**Problem29: Solution2**). Direct Memory Access (DMA) is possible if restricted such as for use by that peripheral only. For example, the BMC cannot issue a PCI Express DMA request into host memory. If that were allowed, a compromised BMC could implant malware, crash the host, or intervene with the BIOS

update process such as replacing the update image contents after it was verified and before being written to the BIOS ROM.

2.6.3 Boot Measurement

Since platform security is typically not a one solution fits all area, additional concurrent solutions can help assure the end user or administrator of intended operation (**Problem30**). As an example, the Trusted Computing Group (TCG)‘s Trusted Platform Management (TPM) Specification defines a TPM chip that securely stores critical credentials and offers provision to perform a “measured boot to detect how and where improper modifications have been made to a system” [28]. Customers are encouraged to utilize TPM for boot code measurement as an additional, complementary security method for verifying the UEFI code (**Problem30: Solution1**). It is complementary since it can notify of a fault but does not prevent bad code execution per se.

Chapter Three: Conclusions

This report contributes to a generally under researched and poorly practiced area of cyber-security in the area of embedded systems as they pertain to enterprise product usage. The report's logical bottoms-up sequence starts from local physical one-to-one attack methods and moves to remote one-to-one and one-to-many software only attack methods, including BMC and BIOS domains.

This report enumerated a total of 30 specific and practical security problems currently observed in real-world enterprise products' systems management and BIOS subsystems. This report also offered a total of 50 practical, and in some cases new and novel, suggestions to mitigate the explored problems. The author felt that this approach creates a much more valuable survey to bound the leading problem areas than exhaustively focusing on a single problem. The additional value of this research was that even though the focus was on enterprise product embedded systems, most of the highlighted issues and suggested solutions apply directly to other industries and product types and regardless of various evolving technologies.

The sources of this practically focused report included online research, first hand experimentation with high volume, mainstream rack servers and interviews with respected industry practitioners and experts. As a practitioner in this area, it is the author's sincere desire that readers can take away practices that can be immediately implemented in their product portfolios and design processes. Interested readers should also now have focus areas to further explore toward bolstering enterprise platform security, while maintaining optimal design for manufacturing and debug with cost effective designs.

Chapter Four: Future Work

Most of the highlighted areas within the enterprise platform cyber-security domain can benefit from future research and work to extend the enablement, adoption and proliferation of best practices at the silicon, circuit board, firmware/BIOS/software and system levels. Solutions that require long lead time solutions and similar multi-sourced solutions, such as silicon impacting or complex software designs, make it critical to stay on top of quickly evolving attack methods. A few key areas where the most value in future work lies related to enterprise products are:

1. To determine how these problems and suggested solutions can be maximally applied to additional industries and applications including the Internet of Things, where physical attacks are much easier.
2. To expand ways to get secure boot enabled in low end general purpose microcontrollers and programmable devices such as CPLDs and FPGAs, which today essentially only check for CRC-based integrity.
3. To explore how BMCs can play a more active role in verifying and recovering, via manual user or automated policies, other active programmable image entities in the system including BIOS, management engines and controllers within commodities such as power supplies, backplanes, hard disk drives, eMMC type flash devices and even more mundane ones such as smart fan controllers, or user interaction LCDs. This includes RAM and ROM limited devices.
4. To explore methods to securely support a 3rd party "application store", giving the server administrator the ability to safely load and execute their own programs into a management controller subsystem without affecting additional functionality.
5. To investigate optimal methods of requiring lightweight authentication to secure GPIOs and memory mapped peripheral local access.

References

[1] "The Global State of Information Security Survey 2016", http://www.pwc.com/gx/en/issues/cyber-security/information-security-survey.html.

[2] "Illuminating the Security Issues Surrounding Lights-Out Server Management", https://www.usenix.org/system/files/conference/woot13/woot13-bonkoski_0.pdf.

[3] "Remote Access System for a Programmable Electronic Lock", US patent number: 5,774,058.

[4] "Hood intrusion and loss of AC power detection with automatic time stamp", US patent number 6,289,546.

[5] IPMI Specification, V2.0, Rev. 1.1. http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html.

[6] "Dell Lifecycle Controller Graphical User Interface, Version 2.00.00.00 For 13th Generation Dell PowerEdge Servers User's Guide", http://topics-cdn.dell.com/pdf/integrated-dell-remote-access-cntrllr-8-with-lifecycle-controller-v2.00.00.00_Connectivity%20Guide_en-us.pdf.

[7] "BIOS Setup User Guide for 13th Generation Dell PowerEdge Servers", http://en.community.dell.com/cfs-file/__key/telligent-evolution-components-attachments/13-4491-00-00-20-44-05-27/BIOS-Setup-User-Guide.pdf?forcedownload=true.

[8] "Intro to Dell PowerEdge iDRAC Quick Sync NFC Bezel", https://www.dell.com/learn/us/en/555/videos~en/documents~poweredge-idrac-quick-sync-nfc-bezel.aspx.

[9] "Filter Driver", https://en.wikipedia.org/wiki/Filter_driver.

[10] "ANT Catalog: Servers", https://nsa.gov1.info/dni/nsa-ant-catalog/servers/index.html#GODSURGE.

[11] "Hacking the Dell DRAC", http://blog.ignoranthack.me/?p=86.

[12] "Owning Dell DRAC for ONE AWESOME HACK!", https://www.trustedsec.com/september-2012/owning-dell-drac-awesome-hack/.

[13] Bonkoski, Anthony J., Bielawski, Russ, Halderman, J. Alex, "Illuminating the Security Issues Surrounding Lights-Out Server Management", https://www.usenix.org/system/files/conference/woot13/woot13-bonkoski_0.pdf.

[14] "ILO default password", http://community.hpe.com/t5/Remote-Lights-Out-Mgmt-iLO-2-iLO/ILO-default-password/td-p/4713914.

[15] "Configure the iDRAC Remote Access Console", https://docs.extrahop.com/current/configure-i-drac/.

[16] "iDRAC7 now supports Default Password Warning feature", http://en.community.dell.com/techcenter/b/techcenter/archive/2013/07/16/idrac7-now-supports-default-password-warning-feature.

[17] "Redfish Developer Hub, http://redfish.dmtf.org/.

[18] "The Heartbleed Bug", http://heartbleed.com/.

[19] ""Bloodsucking leech" puts 100,000 servers at risk of potent attacks", http://arstechnica.com/security/2013/08/remote-admin-tool-imperils-servers/.

[20] "Security Features in the integrated Dell remote Access Controller - April 2016 update", http://en.community.dell.com/techcenter/extras/m/white_papers/20095301/download.

[21] "Introducing "OpenBMC": an open software framework for next-generation system management", https://code.facebook.com/posts/1601610310055392/introducing-openbmc-an-open-software-framework-for-next-generation-system-management/.

[22], "Microsemi Secure Boot Reference Design White Paper", http://www.microsemi.com/index.php?option=com_docman&task=doc_download&gid=133604.

[23], Tsai, V., "e·MMC v4.41 and v4.5 Architecture for High Speed Functions and Features" http://www.jedec.org/sites/default/files/Victo_Tsai(1).pdf.

[24] Wong, Bill, "Pseudo-SLC Flash Provides Design Flexibility", http://electronicdesign.com/site-files/electronicdesign.com/files/uploads/2013/09/FAQs-Toshiba-September.pdf.

[25] "Intel C610 Series Chipset and Intel X99 Chipset Platform Controller Hub", http://www.intel.com/content/www/us/en/chipsets/x99-chipset-pch-datasheet.html.

[26] "System Erase in Dell 13th Generation PowerEdge Servers", http://en.community.dell.com/techcenter/extras/m/white_papers/20440883.

[27] "Unified Extensible Firmware Interface Forum Specifications ", http://www.uefi.org/specifications.

[28] "UEFI Secure Boot in Modern Computer Security Solutions ", http://www.uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf, 2013.

[29] "Intel Boot Guard" https://en.wikipedia.org/wiki/Intel_vPro#Intel_Boot_Guard.

[30] "BIOS Protection Guidelines for Servers", http://csrc.nist.gov/publications/drafts/800-147b/draft-sp800-147b_july2012.pdf.

Vita

This report was typed by Timothy Michael Lambert, who can be contacted at the permanent email address timlambert2@gmail.com. Timothy Michael Lambert wrote this report in fulfillment of the requirements of the University of Texas at Austin Master's in Engineering focusing on Software Engineering.