Version 2.0.3

dan farmer
zen@trouble.org
August 22nd, 2013

IPMI: FREIGHT TRAIN TO HELL
OR
LINDA WU & THE NIGHT OF THE LEECHES

Executive Summary

IPMI is a protocol mainly used to facilitate remote management of servers. Published by Intel and created in conjunction with other major vendors it's nearly universally supported and is widely used for emergency maintenance as well as the provisioning and rollout of applications, operating systems, and various other administrative tasks.

An embedded system called the BMC implements IPMI and lives on server motherboards. It often (especially recent flavors) runs Linux and has its own independent CPU, memory, and storage. The BMC also may provide remote web access along with email capabilities, LDAP support, emulation of remote CDs and other media, and a host of other capabilities. The BMC operates and controls the server at a very low-level, and is mostly invisible to the operating system. Designed to operate even when the server is powered off, anyone who has administrative access of either the BMC or IPMI (they're closely related) enjoys complete control of the server.

There are three classes of serious security problems with IPMI: the Intel specification, the vendor's implementation of the protocol and the BMC, and how it's all used in the wild by the end users. While none of them are individually showstoppers, when combined they create a monumental security problem about as large as a digital Grand Canyon. Unfortunately the issues are complex, involved, and unless one is versed in those three issues the full extent of the overall problem will not be clear. This makes the problems not only even more disastrous, but all the more poignant.

The **IPMI specification** has flaws that allow attackers to access passwords remotely as well as grant system level access without any passwords at all. It also mandates that passwords must be stored or processed in unencrypted form, a very unusual and unsafe practice[1] which means that anyone who compromises the BMC or has physical access to one can uncover passwords used to secure your inner corporate sanctum. Very low-level access to the server and its bus is mandated and may be used to compromise it at a very low and nearly undetectable level.

Vendors frequently rebrand what is, at its core, IPMI – Dell has iDRAC, Hewlett Packard iLO, IBM calls theirs IMM2, etc. – but they also add lots of features, insecurity, and mystery to their implementations, along with enabling some, if not most insecure options of IPMI as a default. The BMC's security is marred by the number of programs and functionality they jam in along with the slow patch times typical of embedded systems. Owners can't fix or patch BMC security problems because the vendors ensure that only their own proprietary software to be used. There is almost no visibility to any of the activity on the BMC, and currently there aren't any forensics or security tools to help with the situation; even backing up the firmware is disallowed. Most servers sold even have vendor placed backdoors on their BMCs so that their support staff may gain the access and control that you cannot. As a final straw: it appears that due to architecture decisions it's possible for an attacker to permanently compromise a BMC, short of some unknown vendor trick or physically damaging the motherboard.

Users tend to manage servers in very large collections that all share the same IPMI password: groups of 100,000 servers or more are not unusual with very large hi-tech organizations. While IPMI doesn't force this behavior both the protocol and vendors almost collude in making it very difficult to do anything else. And because of the difficulty managing IPMI the passwords tend to remain unchanged a very long time, often measured in years, which also makes any password compromise especially serious. *Any* server that is compromised in a group may compromise *all* of the other servers. This risk extends to de-provisioned servers, which might end up on eBay or an auction – they probably still have your passwords on them.

In sum, you may not know it, but your goose may already be cooked and you're simply asking Intel and your server vendors to pass the orange sauce.

Table of Contents

Note: there are so many different implementations, versions, vendors, and ambiguities I've tried to make the writing clearer by putting many of the caveats and one-offs in footnotes at the end of the work and out of the path of general reading.

When talking of vendor implementations I'll often refer to the group I'm discussing as "most servers" or at times "most vendors". I'm referring to either sheer #'s, in terms of market share and what's out there (e.g. since most servers are sold by a small number of vendors, it may be numerically just a handful of vendors I'm talking about, as I don't have #'s or data for a more complete understanding.) At times, however, I'll extend my proclamations to most vendors, since some things seem more universally true across the board. In both cases there may well be exceptions, I'm only talking about what I've personally seen and read.

And while I thank some very important people for help and reviewing my work (see the acknowledgements at the end), any omissions, errors, falsehoods or whatnot are solely my responsibility. It's a sprawling topic, far too big to cover in these pages, but I've also placed some additional background data, notes on vendors and specific implementations, along with links, references, and the like at my web site:

http://fish2.com/ipmi.

I. Background

By far the most popular OOB protocol today is the Intelligent Platform Management Interface, aka IPMI, which communicates to a server via the service processor or baseboard management controller (BMC.) 200 computer manufacturers are currently on Intel's Adopter's List[2] and it's hard to buy a server that doesn't have native support.

What most call IPMI is actually composed of four main specifications: the "Intelligent Platform Management Interface (IPMI), Intelligent Platform Management Bus (IPMB), IPMI Platform Management FRU Information Storage Definition, Intelligent Chassis Management Bus (ICMB)"[3]. It was initially developed by Intel, Dell, HP, and other large corporations; version 1.0 specifications published in 1998. Version 1.5, which introduced networking and is still in wide use, came out in 2001, and version 2.0 was rolled out in 2004 (the last revisions being published June 2009[4].)

Despite version 1.5 being over a decade old an Internet-wide IPMI survey (see section 5) revealed that of the more than 300,000 servers discovered, a whopping 63.1% didn't speak version 2.0. Version 2 does have backwards compatibility, but since virtually every server sold today supports version 2 it's a curious figure.

Like any good spy IPMI has a variety of aliases. Server and firmware vendors not only add a wide variety of features but rebrand the label: Dell calls theirs iDRAC, Hewlett Packard iLO, IBM IMM, etc., but in the end most of the core functionality is powered by IPMI. Intel launched a similar effort for personal computers called Active Management Technology (AMT) that shares many features with IPMI but for a variety of reasons I don't think it's as dangerous[5].

The most common use of IPMI is to monitor server physical health and status via its sensors; temperature, memory errors, disk health, fan speeds, and the like may be captured and associated alarms triggered when something is amiss. Vendors allow you to reboot a server, change BIOS settings, install an operating system, or perform other low-level management tasks.

From the start the BMC was designed to be resilient, and is able to communicate with its server or with other computers even when the main network was down, the server power is off, the operating system or disks crash, or when other catastrophic failures happen (which is pretty cool if you think of it.)

Figure 1, courtesy of the Intel Development Forum, might help illustrate IPMI's place in the management stack. Computer management systems have long tried to tackle the higher-level concepts of applications, operating systems, and the like, and something was needed to deal with the low-level hardware issues. Supercomputers, virtual controllers, clustered computers, etc. also use IPMI to spin up and down nodes on demand.

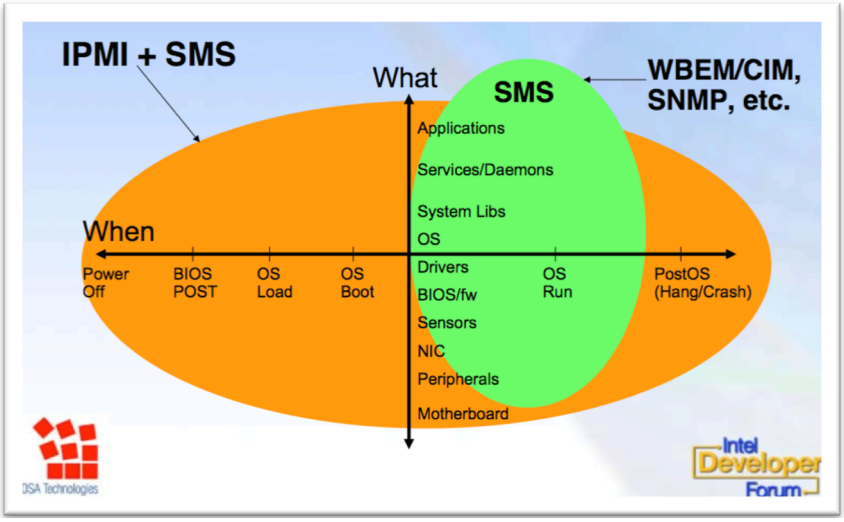

Figure 1

Through efforts such as the Common Information Model (CIM) IPMI is providing its services to higher and higher levels of abstractions in an effort to provide a unified administrative interface. As a result, as well as the lower level tools support is available for most popular scripting languages and web interfaces.

II. The IPMI Protocol

The IPMI specification was written when the performance of embedded systems were an order of magnitude or more less than today's computers, so some of IPMI's design might be understandable in that light. But times have changed, computers are far more powerful, and even when created some of the design decisions are simply deplorable with respect to security. The vendors, who usually turn on all functionality rather than only enable a more restricted and secure set of options, only aggravate the problem.

Even if you're using moderately secure options in your own environment, unless the bad options are explicitly disabled you can still be vulnerable. I've seen many examples on the Internet of experts counseling others on how to "fix" problems by explicitly discarding security controls.

Other than optional password hashing the IPMI version 1.5 has no cryptographic protection at all between client and BMC; the specification defines the RMCP (Remote Management Control Protocol) as a simple UDP datagram driven protocol. As a result it's vulnerable to a plethora of network security attacks such as password sniffing, network spoofing, connection hijacking[6], Man-in-the-Middle attacks, and more.

Authentication has various options, and passwords may be sent in clear text[7] or as hashes, depending on the request and what the client and the server support. When you set or change a user's password the actual password is sent over the network in clear text.

The specification also allows remote information about the security posture to be gained in a number of ways, but the mandatory IPMI "Get Channel Authentication Capabilities" command, which may be done remotely, anonymously, and without authentication, will give potentially dangerous details about a remote BMC including:

- Whether or not the anonymous user is active or not. IPMI defines as the anonymous user as an account with a NULL (e.g. zero-length) name and password. This account gives attackers a free login to a BMC with no authentication.
- It also reveals if the "none" authentication is allowed. This is similar to the anonymous account above, but in this case no password is required while a valid username is needed order to login. While an attacker must know the account name supporting "none" authentication, users will often simply use the default accounts given by the vendor, so it's a dangerous thing to let potential attackers know about (obviously all accounts should ideally require a password to begin with, but people being people…)

The "Get System GUID" command is another anonymous and remote command that, while optional, is "highly recommended" in the IPMI specification (GUIDs are required in the 2.0 protocol). The GUID, or Globally Unique Identifier, is supposed to be a unique number that may be used to identify a system. While this might sound innocuous for attackers or security scanners this is a real boon – one of the hardest problems of network scanning is that of uniquely identifying potential hosts, nodes, or targets. An IP address is a very poor identifier, and having a GUID allows the identification of computers through an otherwise shifting or puzzling network topology. And since no authentication is needed it's simple to send a single network packet to re-identify a

potential target; usually no further scanning is necessary to alert any defenders or alarms.

While it has stiff competition, perhaps the worst problem with the IPMI protocol is something that you must read between the lines to understand – **the protocol mandates either the clear text storage of the password on the BMC or to have it recoverable on demand**. Most modern password algorithms use a hashed or salted and hashed password for authentication. IPMI instead uses a dynamic hash for an authentication code to validate a request:

AuthCode = H(password + temporary session ID + challenge string+ password)

(where H(x) is a hashing function):

The challenge string and session ID are determined by the remote client and the managed BMC prior to this request, and may change with any session request, and without the clear text password at hand this hash could not be calculated. Some BMCs appear to keep the clear text password on "secure" storage that is difficult to read after the bootup sequence, but even then if an attacker has root[8] access on the raw BMC memory (or individual processes) is easily scanned for passwords.

This is simple to illustrate – here's an example revealing the components of the hash using *ipmitool*; the password is meaningless, but the username must be valid (color coding is used to highlight the various regions of the text output):

```
$ ipmitool -I lanplus -U ADMIN -P fluffy-wuffy  -H 192.168.0.69 -v -v -v -v user list

        IPMI LAN host 192.168.0.69 port 623
        [...]
        <<RAKP 2 MESSAGE
        [...]

        >> rakp2 mac input buffer (63 bytes)
        a4 a3 a2 a0 9e 70 37 da 6d 4b e6 a6 d9 df 7b 1b
        29 92 a6 f2 b9 33 75 95 09 dc aa d4 ac f2 1b 10
        af 3b 33 cd e3 50 48 47 35 30 30 31 4d 53 00 00
        00 00 00 00 00 00 00 00 14 05 41 44 4d 49 4e
```

Session IDs	Privilege level
Random: client	User name length
Random: BMC	User name
GUID of BMC	

I wrote a small tool, rak-the-ripper, that may be used to test for this problem.

IPMI 1.5 isn't vulnerable to this attack; in 1.5 to generate an AuthCode a signature is first created, using the data you'd like to authenticate, a random number sent by the BMC, a sequence number, and the signature algorithm type; this is then in turn hashed along with the password and the authentication algorithm type. The BMC may then verify that you've sent the correct data and have used the proper password.

I'm not aware of any manufacturers that don't support IPMI version 2.0 now; most servers have backward compatibility with version 1.5, so for them all of 1.5's security issues are carried forward as well. It's important to note that some vendors only have a subset of IPMI 1.5 features (RMCP ping is the only feature I'm aware that's mandated) – check your documentation or with your vendor for specific details.

And while version 2.0 did add enhanced authentication and further encryption support, it also added some serious security possibilities.

While there are some decent security options in version 2.0, all the popular management tools (such as *ipmitool*, *ipmiutil*, *bmc-config*, individual vendor tools, etc.) use plain old 1.5 communications by default. While it's possible to use cryptography to help secure your communications, there are fifteen different and incompatible cipher suites to choose from (in table 22-19, plus one additional suite for vendor-specific definitions), all with various authentication, integrity, and confidentiality algorithms defined[9]. And while the documentation of most tools suggest using a direct physical console or leverage version 2.0 of the specification to encrypt the traffic when changing or setting your password, all this complexity is presumably why none of the first 15 or 20 results of examples returned by Google's search engine actually **used** version 2.0.

I can only guess it was for performance reasons that the various cipher suites allow "none" as a valid algorithm – that is, no algorithm at all. This creates situations where the data might have integrity protection but is sent in clear text, or a message is authenticated but has no confidentiality.

Of the 16 defined cipher suites there are only three that actually supply all three protections (authentication, integrity, and confidentiality.)

Only one of the three confidentiality algorithms – AES-CBC-128 – should be used; the other two are based on RC4, an algorithm with a troubled security history. Only RAKP-HMAC-SHA1 should be used in the authentication – the other, RAKP-HMAC-MD5 – is once again based on a weak algorithm[10]. Of the three integrity algorithms, only one – HMAC-SHA1-96 – should be used; the other two are based on MD5, an algorithm with serious security issues. The "-96" simply indicates that truncates the 160 bits digest of HMAC-SHA1 to 96 bits. So in sum, only Cipher Suite 3 should be used; if your vendor doesn't support that for some reason, 8 and 12 are better than nothing, as at least they offer all three types of protections.

That said, even SHA1 hasn't lived through the test of time. NIST, an organization in the US government that sets standards for proper cryptographic use for its agencies, wrote (their emphasis) "federal agencies ***should*** stop using SHA-1 for generating digital signatures, generating time stamps and for other applications that require collision resistance. Federal agencies may use SHA-1 for the following applications: verifying old digital signatures and time stamps, generating and verifying hash-based message authentication codes (HMACs), key derivation functions (KDFs), and random bit/number generation[11]". This shows the difficulty of specifying cryptographic algorithms in infrequently updated specifications; unless IPMI is updated at some point we may well find ourselves in a place where all the cryptography specified is ineffective.

Depending on the vendor, their documentation, and implementation, this might be one of the few times I might recommend using the so-called OEM options in IPMI, which allow for the vendors to implement all sorts of things. If you feel like they offer better solutions than the rather dismal alternatives above, go for it.

Carrying all of this to its logical extreme we come to the star of the cipher show – the very first one, the infamous **cipher zero**. It is essentially the un-protocol, with no checks for authentication, integrity, or confidentiality. To authenticate an account under cipher zero **any password may be used, as it will be ignored**. A valid account is required for authentication, but virtually any password will be accepted. Most servers appear to have cipher zero enabled by default, and one of the largest, HP, has apparently never allowed you to turn it off[12] (more details and numbers in section 5.) Examples and troubleshooting tips advising the use of cipher zero as a solution to authentication issues abound on the Internet. Here's a quick illustration, with more details online:

```
# Commment: the command in bold shouldn't work due to incorrect
# authentication; -U specifies the user name, -P is the password:

$ ipmitool -H 10.0.0.1 -U root -P calvin bmc watchdog get
Error: Unable to establish IPMI v2 / RMCP+ session
Get Watchdog Timer command failed
Error sending Get Self Test command

# This really shouldn't work either, but does, thanks to C0.  Cipher 0
# is specified by "-C 0", and the "-I lanplus" says use IPMI v 2.0:

$ ipmitool -I lanplus -C 0 -H 10.0.0.1 -U root -P FluffyWabbit bmc watchdog get
Watchdog Timer Use:     Reserved (0x00)
Watchdog Timer Is:      Stopped
Watchdog Timer Actions: No action (0x00)
Pre-timeout interval:   1 seconds
Timer Expiration Flags: 0x00
Initial Countdown:      15 sec
Present Countdown:      15 sec
```

Another major security issue introduced in version 2.0 was because of the newly introduced RAKP (aka the RMCP+ Authenticated Key-Exchange Protocol), which IPMI uses to negotiate a secure connection. RAKP allows an anonymous user the ability to remotely get the password hash from the BMC. This is an astonishingly bad design, because it allows an attacker to do offline password cracking[13] and break into a remote system by guessing its password, and it's unlike any other system I'm aware of (for good reason!) It doesn't appear to be possible to turn off RAKP if version 2.0 is supported short of disabling IPMI over the channel (for most this would be LAN.)

There is **almost** one small bit of good news with IPMI 2.0 – while in IPMI 1.5 the same password is used for authentication and integrity, IPMI 2.0 introduced what seemed like an effective (if almost never used[14]) protection against a variety of password attacks.

RMCP+ (the plus sign indicates it's the version after plain ol' RMCP) may be configured to use a two-key login where the normal IPMI password is used for authentication and a 2nd key, the BMC or Kg key, is used for integrity – and to execute most IPMI commands or to authenticate to IPMI in general you need to know both keys. The Kg key may be chosen on a per-server basis, so in addition to the basic IPMI password you could add a unique password for each server.

This is a very cumbersome and very rarely used mechanism at the best of times, and there's a reason I used the word "almost"! Even if enabled the vendors have almost completely subverted it to extinction, because when using the vendor supplied web-based GUI, network authentication (LDAP, Radius, etc.), or other forms of communications the additional password is ignored and only the primary account password is used.

There are some additional troubling issues: the specification allows users to use null Kg Key, rendering it useless even if enabled, plus the specification explicitly warns that no "secure, confidential mechanism for installing and distributing user keys between BMCs and remote consoles" exists, which certainly dampens my own enthusiasm. In addition search engines show essentially no documentation, war-stories, or examples of use, which means probably almost no one actually uses it.

IPMI is a flexible specification that allows a lot of different configurations, and like most systems some are less desirable than others, security-wise. This malleability gives rise to complexity levels at the point of absurdity; while several vendors do share OEM'd implementations there is a tendency to add a layer of frosting on top that guarantees no

interoperability. The specification also allows for vendor-specific (and undocumented) tactics for most of the major functionality parts.

As a simple example it's possible for a BMC to be configured with up to 16 different ciphers suites, with session-based or session-less communications, varying privilege levels on a per-cipher basis for each of nine communication channels, all of which may have different protocol types, each channel having privilege limits that overrule user defined ones, a dozen accounts per channel that can vary by name and ID number on each channel along with ciphers, privileges and a variety of other things.

No one can reasonably understand the implications of such complex setups, which only beg all the more for analytical tools. There's a desperate need for security research, articles, software tools, and discussion about the implications of IPMI and all it entails.

Finally, the IPMI specification explicitly mentions its requirements for low-level access. The Master Write-Read command should prove fruitful for further research, as it allows "unfiltered access the IPMB, ICMB, private management busses, and PCI Management Bus. This would potentially allow someone to use those commands to send commands to other controllers or write to non-intelligent devices on those busses."

The specification also mentions the usefulness of System Management Interrupts (SMIs) with IPMI, even though SMIs not officially part of the specification. SMIs are the highest priority non-maskable interrupts on a computer, and when asserted the processors are shifted into System Management Mode (SMM[15]) and the SMI handler – simply a bit of code by the vendor – is free to fold, spindle, or mutilate the server in any way desired in a nearly invisible fashion. The IPMI specification puts it succinctly – SMIs have "full access to system memory and I/O space[16]."

While it's difficult to tell what vendors actually leverage SMIs (several vendors mention it obliquely in their white papers or documentation, and HP mentions it in a patent application[17]), but if they are used in conjunction with the BMC it may be leveraged to do nearly anything on its server; for instance in a paper on monitoring virtual machines on a server paper researchers[18] describe how IPMI was used to communicate to a server's BMC to remotely trigger a hardware SMI, which in turn was used to read the server's physical memory on a periodic basis. For better or worse IBM only released this secret method via an NDA, but presumably other vendors have similar undocumented features as well.

III. Vendors: With Friends Like These….

There is a wealth of information available about the functionality of IPMI, but it's hard to find much at all about how the various BMCs out there work, or their capabilities in all the various vendor implementations. For starters on most servers the BMC is a bona fide server itself with a real OS and some fairly complex interactions with its host and the outside world. But rather than a deep dive – which I couldn't do anyway – this is overview that is based on explorations with my lab machines along with examinations of BMC flash upgrades and servers from other vendors.

From the start OOB was meant to communicate when things went south. Initially wary, users skipped the normal network channels to the servers and used such reliable but slow or pricey technology like analogue POTS lines, X.25, dedicated network circuits or other independent networking technologies; the important part was that it was separate from the main network as to be protected from not only potential attackers, but the same outages that might bring down the network.

As IPMI's popularity grew, users and marketing folks naturally started asking for more features, and even more naturally vendors were happy to supply. As a result more and more functionality has been stuffed into the BMC, with most vendors supply at least a dozen or more different services including web, mail, and SNMP servers that are all outside of the IPMI specification.

There seem to be three or more vendors involved with any given computer's implementation: the chip maker, the one or more firmware adder-onners, and the final server vendor who may add their own functionality and possibly an additional management interface for the end users.

Figure 2 shows an architectural block diagram taken from the specification.

Figure 1-2, IPMI Block Diagram

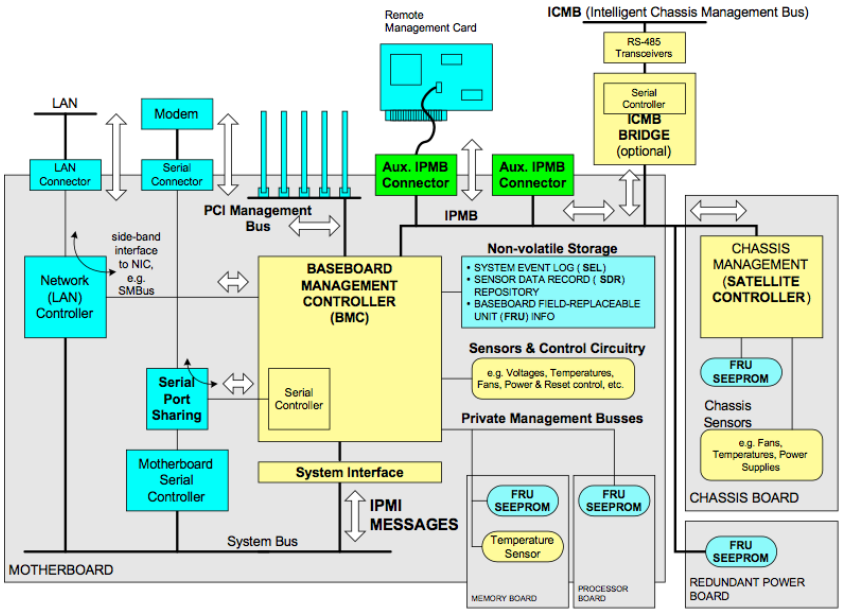

Figure 2

Obviously implementations will vary, but the BMC is a small computer running a minimalistic OS (Linux being common[19]) with an independent CPU (often RISC/ARM-based), RAM, storage, and Ethernet adaptor (although it can share its server's network interfaces as well.) There are just a handful or two of subsystem vendors that manufacture the components that make up the IPMI ecosystem. The major server vendors get their hardware, if not firmware, from Chinese manufacturs[20].

The IPMI specification also defines interfaces used to enable and talk directly to other subsystems such as management controllers, add-in cards, various busses such as SMBus, I²C, and more. It can either directly or indirectly (via its link to the serial console) change BIOS settings and fundamental server configuration settings.

The ARM 926EJ's datasheet, used by Nuvoton's BMC, seems typical (this is powering one of my lab BMCs); it claims that it cruises along at about 200 MFLOPS, which is faster than the first Cray super computer. Since the BMC sits idle over 99% of the time, perhaps we could start harnessing them for SETI or use them as failover servers for light duties. A few hundred million Crays just lying around; perhaps IPMI is simply a missed

opportunity to better our world?

Open source and in particular GPL'd software is heavily leveraged, and from what I've seen typically make up the kernel, OS, boot loader[21], most of the network services, and more[22].

To be sure, the BMC isn't a mere parasite on the motherboard: it's more of a bloodsucking leech, as all it relies on from the server is a very small bit of electricity. It runs independently of the main operating system and is always running as long as any power is supplied to the computer. It's also inscrutable – other than the standard IPMI interfaces used to query its configuration data I know of no software method that would discover activity or details of what's going on within a BMC unless you're logged into the BMC; physical monitoring via JTAG or other instrumentation might prove fruitful. All communication requests are handled by the BMC's kernel and supporting programs, so even the most elemental of requests could return false data.

BMC Communications

There are four main ways of communicating to most modern BMCs directly: an interactive shell (mostly via SSH or telnet interfaces), a web interface, various command line tools (which may be used locally or through a kernel interface to the BMC), and finally via series of network services (e.g. virtual media, remote consoles, SNMP, etc.) that can be used either interactively or via automated tools to manage or query as to the health and well-being of the BMC and its host. Ultimately most, if not all of these capabilities are implemented just like on a normal Linux server: as a daemon or agent on the BMC's OS. Not all of these features are actually in the IPMI specification but are nearly universally on the BMC and packaged up as part of the vendors OOB offering.

The IPMI specification defines channels as communication paths the protocol messages and data flows on. Up to 16 may be used, including the system interface, primary IPMB, and seven additional that are specified by the vendor. The two most commonly used types by users are the LAN and KCS.

Most of the time IPMI must be explicitly turned on[23] via the BIOS/UEFI/system firmware or as part of a special vendor custom order, but it requires no configuration or special software on the host OS to be running (although you have to configure the server in order to communicate with the BMC.) Many servers have a dedicated Ethernet port for the BMC, but it may also share the network adaptor of the host OS[24]. Once a power cord is plugged in the BMC will start, whether or not the host system has been activated.

Depending on the vendor if IPMI isn't enabled there might be a web server on the BMC that can enable it. If nothing else an administrator logged onto the server is able to turn turn everything back on at any time. Most servers don't allow you to turn it off completely.

Authentication

Pure IPMI authentication is handled via a small set (usually from 10-16, although the specification allows 63 per channel) of local IPMI accounts, which may be given passwords of up to 16 (IPMI 1.5) or 20 (IPMI 2.0) characters, and have one of five privilege levels (ADMINISTRATOR being the highest level. Accounts may be disabled or have a NULL (e.g. empty: "") password.

The client either sends the password in clear text or uses a cryptographic hash to authenticate to the server. The BMC usually will usually support network authentication – the usual suspects like LDAP, AD, RADIUS, etc. – I've heard rumors that at least in some implementations that they resort to some sort of hackery (such as simply

appending the plain text password as an attribute, not quite what you might want; in any case it may well be different than what one might expect.) Network authentication isn't in the IPMI specification in any case, so the implementation details are generally unknown. The specification requires at least a single IPMI account on the BMC, which if nothing else it's useful to have a fallback mechanism so you'll have console access during those emergencies when your network or RADIUS or AD servers are down. On the downside, you have that account still active, and it can be exploited.

For non-network based authentication vendors also use the IPMI password as a general catchall for BMC authentication, including web access, their OEM vendor commands, SSH, and for other capabilities.

Most actions explicitly require a password to be entered. However there are some special cases where this isn't true.

If you're logged into a server with an administrative account you enjoy a special relationship with IPMI; you can perform **any** IPMI-related action (including enabling IPMI) on that local server and BMC without any authentication whatsoever. This means that if a server is compromised local IPMI passwords or accounts may be modified, deleted, or created, network services enabled, etc. If cipher zero (0) or the "none" authentication type are enabled any user may be logged into without any password (or any valid one) as well.

SSH and telnet (still alive after all these years) also aren't in the specification, but are nearly universally supported and enabled. In the past they'd grant access to a command line system known as the Systems Management Architecture for Server Hardware (SMASH) and Command Line Protocol (CLP.) SMASH/CLP had to evolve from a committee – i.e. pretty arcane but make a certain amount of sense when looked at it from the right set of angles. Newer BMCs seem to be moving to SMASH 2.0 over WS-Management over TLS.

While it's also possible to store SSH certificates on BMCs to allow password-free logins these may well be problematic from a user management standpoint, since SSH wasn't in the IPMI specification and each vendor implements things a bit differently.

IPMI managers also need to ensure a process is in place to register all locations that have the key along with a removal strategy after a user is terminated or their system is compromised (which would then allow free access to all of the BMCs and IPMI managed systems her key is on.) Some vendors also support single-sign on authentication; if that is hijacked or the user's normal network authentication can be compromised then all IPMI managed servers would be in trouble as well.

The IPMI passwords have to stored somewhere on the BMC's subsystem. I don't have enough data to really know about what the most common methods of storage are, but at least some vendors simply stick the passwords in a file[25] in on the Linux file system, while others at least nominally hide it.

Networking

The IPMI protocol uses UDP port 623[26] for communication, but a BMC generally runs a half-dozen or so network services out-of-the-box that listen to a variety of ports, and has at least as many waiting to be enabled with the web or vendor command line interfaces. Among the usual cast of characters there's web (both HTTP and HTTPS), SSH, telnet, SMTP Virtual KVM/Keyboard/Mouse, Network USB and/or Virtual Media, VNC, WS-MAN, DHCP, SNMP and more – often vendors have a variety of ports for their undocumented special purpose software. And while not listening to network ports they

also have various client programs that talk to the network like AD, LDAP, RADIUS, DNS, mail (SMTP) and more.

Once more the vendors have the ultimate say on what is offered and how it is implemented. Some of these additional vendor protocols use encryption, but many don't, and they almost all require authentication to be sent over the network to use; beware of sending passwords over public networks! I noted in at least one offering (Dell iDRAC 6) I could log into the web interface over port 80, and then when selecting the virtual media options it switched to an SSL encrypted session – FYI, vendors, that's a bit too late, an attacker could have already stolen your connection or password.

Of a particular and perhaps visceral notice is that most BMCs now offer hooks to Microsoft's RDP/Terminal Services and/or VNC for a better view of the server side. Text or graphical screenshots or even video recording console activity are common features (the images or movies are stored on the BMC's flash file system.) Obviously intruders may use these as well to access the server, and without having to play any tricks to reboot or change IP addresses.

Denial of service attacks targeting a BMC are trivial to execute, especially if encryption has been turned on. The CPUs on these things are fairly anemic; they're slow serving up a web page, let alone dealing with lots of traffic or computation. I routinely crashed my BMCs or wedged network services by simply executing legitimate or somewhat legitimate commands. Indeed, the *rmcpping* documentation observes "that some remote BMCs can get 'confused' and delay packet responses if duplicate packets (with duplicate sequence numbers) are sent in succession very quickly."[27]

I think it's worth mentioning that a fair bit of the BMC's capabilities aren't found anywhere but a BMC; virtual media, SOL, sensor data handling, the IPMI protocol, etc., so they haven't had the same level of scrutiny that other exposed code has had. BMCs based on Linux have millions of lines of code in addition, which is typically a mixture of moderately popular open source blended with propriety code from a small number of vendors.

BMC's seem to be generally configured to disallow direct network communication to and from the server, so the BMC can't directly mount network shares, SSH to it's host server, sniff network traffic, etc. without changing configuration. I've had limited success trying to eavesdrop on the other sides but have strong suspicions that someone skilled in the arts (or even semi-skilled, I don't really know much about embedded systems, kernels, or low-level interfaces between systems) could enable the BMC more spying abilities by simply using the command line. Curiously the Dell allowed the server to sniff the BMC's traffic when they were sharing an interface and not the other way around, but this seems unusual.

In the servers I've examined the BMC can see all the network interfaces on the box[28], but are generally prevented from listening to server traffic (and vice-versa.) Given that most operations work fine and some allow network listening, I assume it's a configuration issue; I'm hoping someone will get a more definitive answer, but time will tell.

Finally, like any other Linux system, the BMC may mount any supported types of network file systems, for additional storage, stashing or retrieving data and tools, etc.[29]

Virtual Media and USB

Along with the remote console feature, virtual media is one of the main reasons people really like OOB management. While it's not in the IPMI specification I'm unaware of any

vendor who doesn't offer this feature on the BMC, although sometimes only with their advanced or enterprise version (for an additional fee, of course.)

Using the virtual media feature a user can mount Disk images, USB sticks, DVDs/CDs, and the like from anywhere on the net and they'll appear immediately as a file system on the host exactly like their physical counterparts (beware of autorun![30]) Virtual media is heavily used for provisioning or bootstrapping new servers or applications, remotely installing or reinstalling the OS, deploying diagnostic tools, etc., and uses the standard methods of remote file access (*tftp*, *ftp*, Window shares, etc.) I believe that most leverage a shared USB bus (not necessarily all USB busses!) with the server to accomplish the transparency – I've heard that some recent IBM servers go so far as to use USB as a communication channel for IPMI commands and networking, which might open up some interesting situations.

Virtual media is also perhaps the most straightforward way to take control the server host, or at least the easiest to explain – simply mount a live CD with an OS of your choice, ensure the boot order is correct, and then reboot the system – after all, this is essentially what it was designed for. An ephemeral OS booted in RAM could explore its host server's disks, applications and data, which could be folded, spindled, and/or mutilated unless encrypted. This is fast, as well – an automated attack might only take a minute or three to reboot and scan a server, and before operations explores the any failures of the server an attacker can reboot the server with things pretty much back to normal due to some mysterious failure.

At least some vendors support the encryption of remote media, but both sides of the connection must support it, and some types simply don't support it (e.g. *ftp*, *tftp*, etc.)

Both the server and the BMC may listen to any shared USB hubs, and any virtual media that was encrypted over the network will show up unencrypted on the USB bus.

Video and Remote Console

The server's video memory is accessible to the BMC to allow video recording and snapshots of the server's screen, as well as remote console access. Not surprisingly an attacker may use the same tools available to the owner. Most vendors hand out large Java programs for users to run; Dell (and presumably others, given it appears to be an Avocent binary) has a little command line program that may be run on the BMC do this as well; here's the command and a thumbnail of the image it took.

```
[WPCM450 /bin]$ avct_control capture --file /tmp/snap-from-bmc.png
Capturing screen to file '/tmp/snap-from-bmc.png'... Captured.
```

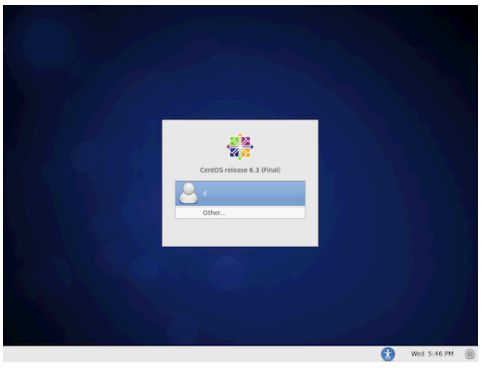

Figure 3

It was also simple to capture keystrokes when a user was logged into the BMC and was virtually "typing" commands to the BMC with the on-screen keyboard.

Storage

All the BMCs I've seen use two or three basic methods to store data: the first is primarily used for the boot disk, often a simple MTD (Memory Technology Devices) chip that uses flash memory without a controller. This is typically broken up into partitions that contain the boot loader (Das U-Boot seems popular) and various parts of the Linux file system – usually the cramfs (which is read-only) or JFFS2 filesystems.

To protect against wear and presumably to make it harder to attack the BMC, one or more RAM drives are created in the BMC's memory and the flash memory is copied onto a read-only RAM disk. You can sometimes remount the RAM-disk file systems to be writable, but any changes only will persist until the BMC is rebooted. When a BMC is upgraded it simply blows away the entire core operating system and writes in a new one.

Many vendors (HP, Dell, IBM, etc.) try to prevent the BMC from running any code but their own. Some use an actual physical jumper that prevents you from writing to the boot block, others use cryptographic storage and/or coprocessors from allowing you to access the raw plaintext key store, and so on (and some do nothing!)

However, there are operations that need a writable store, and some require data to be malleable but persistent, such as with configuration files, activity logs, etc. Upgrades (which usually contain the boot loader and operating system) aren't simply slammed over the existing content – instead the new firmware image is stored in a holding area, usually checked for validity and possibly a digital signature, and then moved into the area where the last file system data lived.

Since the vendors don't allow you to backup your BMC firmware (only the basic configuration), they'll typically have room for a version or two of past flash versions, so if the newly flashed one spits up a hairball you have a chance to roll it back to a working copy. This leaves a lot of space for data– in the tens if not hundreds of megabytes.

Flash disks, like most storage medium, are also vulnerable to even simple data recovery methods. I conducted some tests using very crude techniques and was able to recover about 2/3 of the free space I'd written to with the lowly (or mighty!) *cat* program on the raw device. It was easy to recover data written to disk after removal, even after pulling the plug and waiting 30 minutes to ensure the data should have been gone for good.

More information on the results and techniques are available online, but this wasn't very surprising given earlier work I'd done on the difficulty of destroying information as well as other people's work[31] on recovering data from flash memory. Forensics research has clearly shown that data is often fairly difficult to eradicate; since IPMI passwords and hashes appear to be frequently stored there it seems as though even if the vendor's supplied methods – if they exist – take substantial precautions that the passwords could still be recoverable from the BMC's storage. In addition there was also some, although substantially less, memory retention in the BMC's RAM over cold boots with similar types of testing.

Internal BMC Security and Trust Architecture

There are some security architecture problems with the way BMCs are architected that seem relatively universal among implementations. It shouldn't be surprising that it's simple to access any and all communications to and from the BMC, of course, including anything on the web interface, USB, network, etc. Encryption won't help, since the BMC decrypts it anyway, so a compromised BMC could easily be used to capture IPMI passwords, as well as any session or virtual media information, including any passwords and locations that the BMC uses to connect.

By far the worst, however, is that an attacker can create a persistent compromise. And when I say persistent I don't mean simply continuing through a power cycle – I mean that unless you know something I don't I don't know of any way to kick them off the BMC. This requires a little knowledge, but I can only assume it'll show up in exploits soon.

The vendors may have a devious way of getting in that doesn't require the BMC's participation, but it could be that physically damaging the chip and kill IPMI altogether is the only hope. JTAG – if supported at all – would probably work in the right set of skilled hands, but no public documentation exists on using that on any BMC that I've ever seen.

Such a persistent break can happen (at least on Linux-based models) because the BMC is simply a Linux server, and all the configuration changes, IPMI commands, flash upgrades, and changes you might want to make on it depend on programs and shell scripts that can be modified or disabled if an attacker has compromised the BMC, leaving you incapable of managing your own hardware. And even if you had root access before, you may not for long.

I was able to do this on a Dell and SuperMicro BMC. It took little effort to place two types of persistent data on their writeable file system partitions: the first was only semi-persistent, and was simply placing SSH keys and configuration changes that allowed remote host-based administrative access with no additional authentication; the second was a program (shell script) that executes during the boot process with system privileges.

Both are (a) invisible to the owner of the machine[32] and (b) allow persistent, long-term access if desired. I didn't completely shut off access to the system because I wanted to retain an entry point for myself(!), but with a little more effort one can imagine a system that would simply wait for external commands before springing into action.

If you had access to the shell of the BMC it'd certainly be possible to eliminate the SSH hole (assuming you could find the modifications); re-flashing the firmware would probably remove the changes as well, but it depends on vendor implementations.

However – if you change the boot process, which I did in the 2nd type of compromise, it means that I can do anything I want. I was able to kill off the BMC agents that were listening to outside commands, making it, as far as I could tell, impossible to dislodge me. This is because all inbound BMC communications go through pretty simple channels – a daemon or agent here, a web UI that makes an API call or executes a program there, etc. As an attacker I can, and did, disable all of these. Short of physically damaging the chip, zapping the storage somehow, using JTAG or some vendor trick doesn't depend on the BMC itself to be trusted to do the right thing – **I don't know of any way to get me off the system**. This works because nothing operates at a lower level than the BMC; any attempts to update the flash, via the BIOS, CD, vendor, command line tools will result in the same thing: they all ask the BMC "would you please update yourself?" This is logically consistent with the design, at least, because vendors don't want you putting your own firmware on the system; instead of slamming in new firmware they put it aside to examine it and then decide on whether or not to install it. If I'm in charge I can say no. And even if it's impossible to modify the file system (which I seriously doubt, but I ran out of time!) to change the boot process entirely, as soon as I can execute a program I can instruct it to cease running programs I don't want and then it's game over.

There's a near total lack of visibility about what's going on down in there. About your only hope is external to the box – if an attacker sends network traffic and you're sniffing it you might see something, although it might all be encrypted or obfuscated.

It seems pretty clear that persistent compromises will lead to ads, spyware, and other malware to be injected via the BMC. While it might be amusing in some sense to have to do ad click-throughs to access your remote consoles (or perhaps a simple "deposit 1 bit coin to access your server" for the next 30 days), I don't think many server owners would appreciate the humor.

In the upcoming paper "Illuminating the Security Issues Surrounding Lights-Out Server Management" A. Bonkoski, et eviscerate various SuperMicro server models and tear them into little bits, noting that they could break into forty thousand BMCs in an hour or less with one exploit, and confirm the existence of many more. This boosts the credibility of some predications I made in the last version of this paper, when noting the very poor defensive strategies and code that permeates the BMCs I've encountered. The bad code, crashing and flakiness, developer files and tools left lying around, programs and configuration files that are referenced but don't exist, repeated log entries of missing or broken elements (that are never looked at, since only someone logging into the BMC can see them), all point to poor process and execution – and – more to the point, an indicator of security problems that are simply waiting to be unveiled.

And who knows how long the security bug list that the vendors know about, or the people who wrote the code? I would imagine an author of some of this code might well be a formidable enemy. After finding an undocumented feature on my Dell server[33] that would enable shell access on the BMC among other serious and long-lived flaws it makes me wonder: what else is out there on these black boxes?

And it appears that the vast bulk of BMCs are manufactured in one country – China. I've personally nothing against China in particular; it'd be of similar concern if any single country had such control over such a sensitive piece of technology that is such a sublime possibility for spying and espionage.

The cracks are starting to show – I'll further predict a rather sizeable wave of exploits will be coming out soon. They may already be out, but rather than warning the public of the dangers, they may be simply sold to the highest private bidder to become a part of an electronic weapon. These sorts of exploits are low hanging fruit.

IV. How IPMI has Been Used

Other than the people who actually use IPMI there aren't a lot of people who know what IPMI even is, let alone how it's actually used in the Real World™. Given the flexibility of technology there can be vast differences in implementation, but I feel that there are some common threads.

IPMI generally plays a substantial role helping the basic role of system administrators (and all the assorted and sundry varieties): minimizing costs while maximizing availability and resources to their users. In large part it does this by giving close-enough-to-physical-access while keeping skilled network, host, and application administrators at their desks or at home instead of driving once more to the data center to reboot a system or troubleshoot a problem.

Vendors clearly state you should keep IPMI traffic on its own separate network, but for cost reasons this is rarely done (I've never actually seen it or heard of anyone who actually does this, but presumably some do somewhere.) Instead, as previously

mentioned, either VLANs or non-routable networks are heavily leveraged (and sharing space with all those systems that few ever interact with but are pretty crucial to running a network or datacenter). System administrators know that IPMI is a loaded gun, so intentionally placing an IPMI interface on an unprotected network segment or the Internet is not the norm, but as evidenced by the recent survey data (see section 5) it still happens a fair bit. Remote access to IPMI networks are instead protected by some of the highest security in the organization, via VPNs or other network choke points that require strong authentication to enter.

To do anything with IPMI you must be authenticated[34]; in my experience network authentication isn't commonly used – instead a single password (or a small set of passwords) is shared for large blocks of computers.

Network architecture is different than a usual design as well; for instance a typical network zoning architecture might look something like figure 3, partitioned into network areas by routers, load balancers, firewalls, and other network gear in or between each zone.

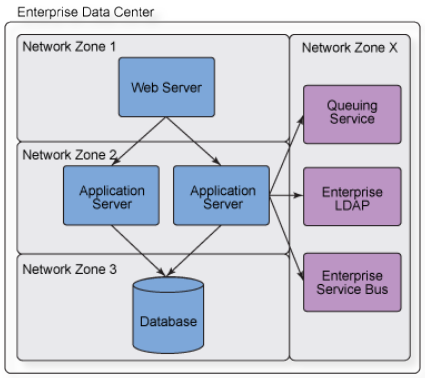

Figure 4

IPMI networks, on the other hand, are generally grouped by geography or by system management or provisioning groups rather than by the usual network topologies and security groupings of application or service offerings. In the image above all the servers might well in a single IPMI group.

This is an important distinction, and I'll repeat – IPMI groups are generally duplicated throughout blocks of servers that are aligned with geography and operational groups, not by the server's function, business owners, and the like. While they're nothing stopping anyone from using a variety of methods to manage the passwords, it's often done this way because the same people managing IPMI are not the same people managing the higher layers of the computer, plus it's the easiest way to deal with a complex problem.

The IPMI network, rather being segregated or split into security zones, is usually one big flat area – it transcends the usual network and security architecture and affords unfettered access to large amounts of important systems that cut across organizational boundaries. This is by intent, and the idea is that no one but very special and trusted people should be mucking around back there. To make things more interesting the usual security measures and network monitoring usually isn't done in these back networks for the same reason they're shared with IPMI in the first place – it's prohibitively expensive implement, and the real threat is thought to be outside of the network[35].

Large organizations – especially the more modern or tech-savvy ones – often have very large IPMI groupings of managed servers; 100,000 or more are not unheard of (IPMI is especially important for bootstrapping installations, provisioning and maintenance.) This isn't really new: the computer-as-a-utility model means huge data centers that stamp out lots of servers, and running internal or external clouds is pretty standard these days. In these larger groupings IPMI plays a big role in provisioning via PXE bootstrapping and fast provisioning in particular. The IPMI passwords and configuration are usually set when the machine is initially provisioned or via vendor special custom builds at the factory.

While most server vendors have tried to sugarcoat the pig to ease system administration burdens, most server vendors aren't hailed for solving enterprise-level management problems; they are also often active in their efforts to create products that don't interoperate with competing vendor solutions. As a result any substantial heterogeneous IPMI customer implementation generally stick with a minimalistic core of features that can be automated or used in a somewhat cross-vendor fashion, rather than fully implement all the rich vendor sets of features.

Another this-doesn't-help-IPMI is that no one really knows where all those servers are. As amazing as it might seem to those not in the server management business, actually knowing just the basics about a random server (e.g. where it is, what it does, who the business owner is – heck, if it even exists) with, say, 90-95% overall accuracy is doing a tremendous job. There's an entire research field and sets of solutions (that don't work very well) dealing with the problem of building a somewhat Sisyphean configuration management database (CMDB), in the hope that you'd have an Oracle that knew all. Across large enterprise the numbers get even bleaker. This in turn means that changing IPMI passwords is very painful and rarely done. The usual network based user and password management tools don't work, and changing **most** of the passwords doesn't cut it: if you miss any you're forced to have to keep track of the older passwords used as well, in case you run across a system that was missed in the last round of changes.

Essentially the password management situation is using static configuration files in a large dynamic situation – horrors such as the old "HOSTS.TXT" files and its friends are the reason we moved to DNS and network authentication all those years ago; even 30 years ago we knew that static just doesn't scale or work in large networks[36].

Most organizations of any size have a small group of trusted individuals that know the IPMI password or how to get it (e.g. a physical safe or lockbox.) If one of the anointed few left the organization you'd just kind of hoped that they wouldn't be evil rather than go through the monumental pain of change. Plus the general feeling seems to be that they'd have to have inside access to the datacenters to have it make any difference, as the server's IPMI network interfaces weren't exposed to the general populace.

In reality IPMI passwords aren't available to a few trusted souls, however. Management consoles and automation scripts have them embedded in configuration files or in the code. Mobile IPMI management apps are flourishing and routinely save the password on the device. Browser caches on administrator machines are another good place to look. Backup servers have a wealth of passwords stored on them, often spanning different IPMI domains.

Mistakes happen, as well; I had a good source inform me that a Fortune 500 company accidently posted their IPMI password that was embedded in some software to Github (a popular web-based software development site) – a password that was used to manage

over 100,000 servers – and was then quickly and quietly pulled it back. As far as I know the password is still being used.

In sum the current security strategy is to only allow the small and cool group of kids the magic password and try to keep the attackers away from those interfaces at any cost. Such things are hard to measure since no one is talking, but I'd guess that the lifespan of IPMI passwords in larger IPMI groups could easily be measured in years.

V. BMCs gone Wild

Other than anecdotal evidence not much is known about how IPMI is actually used, configured, and secured. To gather some data a "Get Channel Authentication Capability" packet was sent (IPMI requires an answer to this request) to all IP addresses on the Internet (e.g., 0/0 minus private networks on the IPV4 space) over an approximately 24 hour time frame. These initial results were used to send a few additional probes to answering hosts. Keep in mind that there are orders of magnitude more servers behind firewalls, gateways, and other network barriers that won't speak to strangers on the Internet, and obviously within any organization there are many more systems running IPMI. In any case 312,357 IP addresses responded to the initial scan, with 295,364 replying with a valid IPMI response packet.

Version-wise, about 2/3rds of the replies indicated that 1.5 was the only one supported, which is surprising, given the age of the protocol (it was last revised in 2004, and version 2.0 seems ubiquitous amount current vendors.)

The fine SSL observatory software was used as a follow up on hosts that replied to the initial packet; since most BMCs support a web interface I thought it'd be interesting to see how common it was to leave it exposed as well. Of the close to 300K initial servers 111,090 were also running a web server with a certificate on port 443. Many scans and surveys have been done about this already, and the certificate news was about as bad as everywhere else. Almost exactly were self-signed (49.8%), and lots of errors abounded, with a full 40% of expired.

Of note were the number of duplicate certificates – many vendors simply use the same certificate for all their BMCs of a certain make and model, and expect the user to change them. They don't. Other than a pair of generic replies with no vendor (e.g. VeriSign and GeoTrust), here were the top ten:

Number		% of Total	Issuer
23506	*(16092)*	21.1	Super Micro /CN=IPMI/emailAddress=**linda.wu**@supermicro.com
	(7414)		Super Micro /CN=Linda/emailAddress=**linda.wu**@supermicro.com
8763		7.9	Aten /CN=doris/emailAddress=**doris**@aten.com.tw
6920		6.2	??? RSA Public Key: (**512 bit**)
5413	*(3577)*	4.8	Dell /CN=**iDRAC6 default certificate**
	(1836)		Dell Inc /CN=**iDRAC6 default certificate**
4531		4.1	Peppercon AG
4487		4.0	GeoTrust DNS:*.**securesitehosting.net**
2347		2.1	??? RSA Public Key: (1024 bit)
1870		1.7	GeoTrust DNS:*.**securesitehosting.net**

My money is on **Linda Wu** has the real power behind the IPMI throne, with Doris a distant second; since the scan only did a small fraction of the total servers in the world,

these two women must have many millions of certificates that they've signed running around. Server certificates are an important front-line defense against man in the middle attacks; duplicate certificates means that if any server of a similar model is compromised, yours or someone you've never heard of, it can put your own users and servers at additional risk. My advice? Don't fight Ms. Wu in a dark alley.

I also did a quick check to see if Cipher Zero was enabled with a subset of the total servers scanned. Of the systems that claimed to support IPMI 2.0 and I was able to determine a valid account, I found that over half had it enabled. While difficult to draw too many conclusions from this it's pretty clear that if people were aware of the problem they wouldn't have this turned on!

Cipher 0 Enabled	Total Scanned	Percent Enabled
20513	35494	57.8

As a final check I used the RAKP remote hash recovery method discussed in section 2 to see if it'd prove to be a reasonable attack method. 29.3% of the passwords were cracked in a modest run of the the Hashcat password guessing engine; the top 10 most popular passwords:

#	Password		
11592	ADMIN	890	superuser
2808	admin	786	computer1
2703	PASSWoRD	567	changeme
2139	calvin	380	4rfv$RFV
972	root	303	password
890	superuser	256	123456

Not many surprises here, although I'll note that about 1/20 servers still use their vendor default password (the top guessed password, "ADMIN", is from SuperMicro.) I'm be putting up more details and additional results (although no individual system's data will be released) on my site.

VI. Security Proclamations

Certainly it's bad enough if someone compromises a server's BMC, because it's meant to be an opaque box that can't be monitored, plus it has a lot of power over the server it resides on. But the real win is to get an IPMI group password, because that can grant access to large blocks of servers in a very stealthy fashion. It's a real killer because in addition to the damage that can occur you usually will never spot it leaking out, so you can't be certain who knows the password.

Unfortunately the BMC server is lacking some of the very basic security controls we now take for granted in modern servers – if someone tried to install a server with equivalent security and operational features on a production or server network zone they'd be laughed out the door.

A basic security lemma is there is no meaningful security without a way of validating the existence of problems or the lack of the same. Typically this is what audit tools do – they allow you to verify security via automation or manual examination. The BMC is almost completely un-auditable, and a crafty attacker may corrupt the results as well.

In addition to its lack of auditability, there's no (legitimate, or at least vendor-supported) chance of performing internal configuration, integrity, and systems testing, let alone change management; there's almost no activity logging available, it's impossible to install 3rd party security or defensive software on the BMC, no host-based firewall[37], no method of enforcing or checking password complexity & strength, default there are administrative accounts with well-known or no passwords at all that are extremely difficult to impossible to audit, there is no documentation, no means of backing up the system, and on and on and on and on. But there are more issues, actually lots more, and in some cases rather unique that add up to real trouble:

The IPMI protocol requires vendors to implement serious security problems.

It really bites to have to use a protocol that can be used against you, because it can take a loooong time to fix, if ever. Some specific issues:

1. The reusable passwords used for IPMI authentication are stored on the BMC in **clear text in flash storage or in its memory**[38]. It can be more-or-less difficult to extract or capture the passwords, but there are numerous ways that you can gain access to them. And most definitely if you have physical access to the server[39].

   This has a bad side effect: de-provisioning systems is really dangerous now. If an attacker can recover the IPMI password from a server it makes the end-of-life process for a computer a bit trickier than usual. The typical best practice for end-of-life-ing a server, even in very high security organizations, is to melt or shred any disk drives and to try and get rid of the carcass. However the IPMI passwords stored in flash are still on the motherboard. Remember all those 2nd hand servers you let employees buy or take home or the old dinosaurs sold on eBay when you did the last hardware refresh? They have your passwords on them still. The same one you use for all the production servers.

   With no documentation it's hard to say how to erase the passwords from the BMCs flash memory – where is it stored? Is it backed up somewhere? Can an attacker recover them with data recovery methods on the raw media, as my tests seem to indicate? Presumably different vendors implement this very differently. I know of no best practices, standards, or even guidance from vendors on how to ensure the password is really vaporized. **I'm not sure if any method other than physically destroying the motherboard should be trusted**.

2. Password hashes may be recovered remotely by anyone who can speak the IPMI protocol. This means attackers can grab your password hashes, run them through password guessing engines offline at their leisure, and then come back to hit you with the appropriate password.

3. Accounts may be logged into without authentication, thanks to our friend cipher zero, the anonymous account, and the "none" authentication method.

4. Information leaks from an overly friendly protocol. It's great that the folks who designed IPMI were concerned about the user experience[40], but you really shouldn't give out information such as password-less accounts exist, whether or not you can login anonymously, and a handful of other issues.

5. IPMI's design requires the implementation to have very low-level and potentially dangerous access to the lowest-levels of a server. And suddenly low-level Denial of service and strange hardware attacks become easy, nasty, and very real. DOS attacks usually bring forth a yawn, but these are a horse of a different color. What if your

server's drives, memory, CPUs, etcetera start disappearing intermittently or permanently? Or if your server's memory or disks get corrupted … occasionally. Or worse.

6. Having IPMI access also means (remote) console access; this in turn grants access to the BIOS or UEFI server configuration via the vendor utilities (usually obtainable by hitting a special key (DELETE or whatnot) right after the boot starts, which can be monitored in via IPMI.[41])
7. Virtual systems can be chewed up from the inside. Of course this isn't a great surprise given all the rest, leverage IPMI to make an IBM server's BMC to facilitate SMI interrupts that would halt the physical server briefly in order to check the integrity of the hypervisor; it also gives an idea of some of the power that servers have by design (and by proxy IPMI) to manage live virtual systems. All of this happens in milliseconds and is generally unnoticeable to the user.
8. If a server or the BMC web interface is compromised an attacker can talk to (or change) the BMC network interface and all of the services it runs on the trusted network[42]. This is because the IPMI protocol (via direct commands or the web-based GUI) grants the power to modify or configure the server's physical network interfaces.

User behavior causes issues as well; they're largely driven by the specification and what vendors give them to work with, but in any case:

9. If you have the IPMI password for a single system, you have the password for all the computers in that IPMI managed group of servers, which are often very large groups. I'm unaware of any implementation that can show how long the password was deployed, last changed, or which systems share them.
10. **Avoid bleeping IPMI mobile apps.** They now have apps for mobile devices that store the IPMI password, and everyone knows how often phones get lost, stolen, or left alone and defenseless in a hotel room. If you value your systems. I find it difficult to agree to any risk calculations required to justify this, unless it's part of a two-factor authentication system and at least one of the factors isn't on that phone!
11. The BMC's ability to leap across interfaces and access anything the server its on is plugged into illustrates a shortcoming of the standard models of network architecture. The back networks that aren't usually exposed and require two-factor authentication, traversing bastion hosts, and the like are actually at risk whenever a system is compromised that has a network interface to it, no matter what network it's on. And shared networks with 3rd tier servers can also grant access to the super-valuable areas you really don't want people mucking around with[43].
12. Given the frequent problems uncovered, systems exposed on the Internet, and a lack of basic knowledge of IPMI security issues, uneducated users are shooting themselves in the foot repeatedly with a large caliber pistol.

And of course **vendors** have introduced their own set of issues:

13. Your servers have IPMI and the BMC functionality even if you don't know it or use it, and it cannot be removed. To be fair most – not all – vendors ship servers with IPMI disabled unless you specifically request it. You can try disabling it, but I don't know of any way to permanently disable via software, and it can be turned back on at any time. Remember, unless power is completely removed from the server (cord detached!) the BMC will continue to hum along, irrespective if it's used or if the host is on or not. Higher level vendor implementations like iDRAC, iLO and the like can

at times be uninstalled but the BMC is usually embedded on the motherboard itself, ready to be flashed and set into action at any time.[44]

14. If you can get any one of these three items – root access on a server, have an IPMI administrative account, or shell access to the BMC, you can compromise the other two.[45]

15. What you can do as an administrator or **root** on an IPMI system is substantially worse than having root on its host computer. In addition to having complete control of what goes on in server-land the BMC is able to manage or control pretty much anything – hardware, software, firmware, etc. – on the computer

16. Controlling the BMC allows traversal of separate networks you might not want to see or think about being connected. It seems to be generally assumed, or at least hoped for, that IPMI and host network access were separate and neither could access the other's network(s)[46]. This is clearly false.

17. The BMC runs a variety of network services that are ripe for security vulnerabilities. By default a half-dozen or so are on out-of-the-box, but others may be turned on and a dozen or more total network services are pretty common[47]. All the programs are fairly old[48] and vulnerabilities are presumably shared for all the similar servers you've deployed from that vendor. While you can turn off some of the services others can't be (like the web server that allows you to configure the system.) And even if disabled an attacker who compromises the server can turn them back on to attack.

    Since I released the first version of this paper more cracks, bugs, and holes are starting to show up. There will be more.

18. When vulnerabilities are found in the BMC you can't apply a security fix yourself because you don't have BMC login access and vendors generally disallow flashing a patched ROM image of your own; you must wait until the vendor puts out a release. It's hard to imagine a much worse situation – even if you disable a vulnerable service and attacker can turn it back on, compromise your system, and then turn it back off again. This is a very difficult attack to notice, let alone stop.

19. The BMC can monitor the host computer, but the host server has almost no visibility to what's going on in the BMC[49]. Not much to say here – the BMC has the all-seeing eye, while the server can't even tell what version of firmware that the BMC runs, given that it has to believe that a potentially compromised BMC will answer truthfully.

20. It's hard enough to tell if a server has been compromised. It's nearly impossible to tell if the BMC is[50]. Ironically the vendors, in presumably trying to protect the IPMI passwords from compromise, have made it very difficult for legitimate users or owners from examining the BMC very difficult; you usually can't detect anything about the BMC's status or state other than the vendor version number, which, of course, is communicated to you via a secret program running on the BMC that an attacker might modify. I ran out of time trying to get the Qemu emulator running on a BMC, but one might imagine a BMC running a virtual BMC and how hard it would be to figure that one out. Dell does have the undocumented "ap-req" command no their BMC, which says "Start DRAC simulation" upon execution, but I've yet to figure out what it really does. Turtles all the way down.

21. Man-in-the-Middle attacks and other cryptography issues are probably not high on your list of concerns compared with everything else, but it should be pointed a fair bit of the traffic going to-and-from the BMC aren't encrypted or protected against this

type of attack. If you're merely using IPMI behind the scenes in your own trusted network it might be fine, but BMCs will shift from encrypted traffic to unencrypted fluidly and without much warning, and some types of traffic is never encrypted[51].

22. IPMI data consumers have to worry more about unfiltered input. Interfaces to the BMC and IPMI have filters to try to prevent dangerous (e.g. overly long, shell meta characters, JavaScript, etc.) data being placed into them; an attacker who controls the BMC, however, can generate arbitrary and unexpected responses. Data-driven attacks, especially when they come from unexpected sources, are very effective. For instance a programmer reading the specification knows that usernames cannot be longer than 16 bytes, and might not anticipate getting back thousands of characters. Popular consumers of IPMI sensor readings, such as Nagios, Zabbix, etc, have already fell prey to data based attacks on other sources of data and have security advisories in their names.
23. Evil maid attacks (EMA). A term coined by Joanna Rutkowska, an EMA is when an attacker has physical access to a computer and can muck with it. If an attacker can replace or modify the BMCs firmware then that firmware can not only compromise that system, but if the maid is able to read the IPMI password…. At the time TPM or other secure boot mechanisms were thought to be protection against such attacks, but IPMI is rock to secure boot's scissors; the secure boot people freely admit that they can't safeguard against physical access – or the equivalent.
24. At least on some servers both BMC and the server can sniff, monitor, manipulate, etc. each other's network traffic. Sniffing traffic can sometimes compromise IPMI keys, and may also inform attackers where trusted systems are (such as those holding SSH private keys, or monitoring or automation servers) if they want to attack upstream or the management servers[52]. I don't have any good data to know how universal or rare this is.

Even if you manage to have everything locked down and make it very difficult to gain access to the IPMI magic password it only takes one mistake or leak of information to grant access to all the servers in the group. Worse still, unless you're very lucky you'll never know anything is amiss.

All of this makes **known** incidents very troubling as well. What is the correct response if you suspect that a server has been compromised? If the IPMI password can be captured by merely being root on a server then the usual defensive strategy of having a small group of trusted people with the password isn't valid even if you don't have an outside attackers – any administrator or legitimate root user now has the ability to get the password.

How to responsibly deal with BMC vulnerabilities is also troubling: what's the correct response if you find a security flaw is found for a BMC or set of vendor BMCs? Today for your average vulnerability it's rather common to send it to full disclosure lists along with a security advisory, often with exploit code or details that leave little to the imagination. Is the calculus here seems a bit different; users have very little recourse in the way of safeguarding their systems against a new exposure since they can't patch their own systems or turn off the problems. Sending the problem only to the vendor is one possible option, but traditionally researchers and full-disclosure advocates complain that vendors are often slothful when there isn't any public pressure being placed on them. There would also presumably be substantial financial incentives to not disclose to the vendor – if you discover a zero day exploit[53] against a BMC you may well wish to sell it to interested parties.

The scale is very different as well – it's very rare when a systemic, vendor-wide, issue comes up for **all** computers from a given vendor (and in this case it could involve multiple server vendors if they share some of the same code or firmware manufacturers.) Coordinating patches fix across all vendors in enterprise would be extraordinarily difficult not only because you don't know where all your servers are, you also don't where the specific vendor types are as well. Operational risk would be substantially higher when trying to deploy new firmware enterprise wide than fixing an application here and there; and of course you should remember that firmware management is typically implemented using IPMI.

In any large organization legacy systems abound – they might run the old payroll system or do some other crucial task that would cost millions to replace. What do you do when a new BMC vulnerability that exploits older systems that aren't supported by the vendor – or the perhaps the hardware vendor doesn't even exist anymore.

Finally – even if you don't care about this whole password thing; if you thought Stuxnet was stealthy, at least it was running in on your CPU – how about something that has full access to your system that's just about impossible to directly discover (let alone analyze) from the OS? For a long time malware has been digging deeper and deeper into the hardware layers, the future of serious spyware and malware will continue to dive more deeply.

In sum: Imagine trying to secure a computer with a small but powerful parasitic server on its motherboard; a bloodsucking leech that can't be turned off and has no documentation; you can't login, patch, or fix problems on it; server-based defensive, audit, or anti-malware software can't be used for protection; its design is secret, implementation old, and it can fully control the computer's hardware and software; and it shares passwords with a bunch of other important servers, stores them in clear text for attackers to access.

Not to mention it was designed for full control, remote management and monitoring, and it's pretty damn good at it.

Hirudo medicinalis
aka
Leechus Amongus

Figure 5

VII. Into You Like a Train (Conclusion)

While I may have painted a bleak scenario, the confluence of issues in the IPMI specifications, the vendor add-ons, and how IPMI is used in the real world really do make one nasty brew. Just thinking about the huge system outsourcing shops and managed system providers with IPMI access to all of their various customers makes me rather ill. The current situation of vendors having black boxes that can wreck havoc on an organization's security so readily seems ripe for exploitation. I don't think it's reasonable to expect that two tightly physically coupled computers that share resources won't leak information or allow one to fold, spindle, and mutilate the other.

I was much more unsure when I started working on IPMI security but since releasing the first version of this paper along with some simple tools no one outside a vendor or three has disagreed with anything I've written. There still is a lot of murk about how IPMI is actually used and deployed, but in the last half-dozen months I've uncovered more serious security problems and the overall situation isn't getting any better.

While IPMI and BMCs have a near breathtaking amount of structural, architectural, and procedural security issues that are really serious, there doesn't appear to be any evidence that many on either side are taking note, and as far as I know, no one has written any

really nasty BMC exploit code that would take over a server and bend it to its will. Of course who would actually know it existed unless it was found? Using a BMC as an attack or spy platform could easily be done today; at least for some number of boxes it'd be pretty simple, even for me.

Leveraging the BMC for the more nasty stuff, like grubbing around in memory, the raw disks, and all that would probably take a kernel or an inside BMC developer to do well, and may or may not be possible on any given vendor's implementation. But for the BMCs that support these operations it could be an absolutely devastating attack.

A further downside for the white hats is that (a) given the small number of BMC firmware manufactures that are widely OEM'd, such exploits cover a very large number of servers in the real world, and (b) such code would probably be modestly reusable. The question isn't how much an attacker wants to spend to compromise your servers, but a given BMC or firmware add-ons in general. High stakes and green-field opportunities seem to be abundant.

Cryptography won't help much. TPM or other trusted boot mechanisms aren't commonly deployed (and especially not for servers) and they also explicitly state that if someone has physical access – or the equivalent, in IPMI's case – all bets are off. Indeed, sniffing, inserting, or modifying cryptographic keys isn't incredibly difficult if you can modify the basic boot process as well as monitor and change system components and capability at will.

And of course it's not quite time to stick our head's in the virtual oven. There are a slew of 3rd party OOB, IPMI, and system management systems out there that might be leveraged for better security. Of course even if they do work firing up a large management system always takes a lot of effort to implement as a customer, and even if successful the management systems themselves become a very large and valuable target.

HD Moore, of Rapid 7 and Metasploit, seems to be one of the first to take this seriously and start writing some auditing and testing tools for IPMI and BMCs[54].

We are a bit behind the eight ball right now – there doesn't even exist a reasonable set of technical security and configuration best practices, which is pretty amazing. Other than my own advice I've yet to run into any checklists or best practices other than "ensure the IPMI interface is on a segregated network that is protected by a firewall", and "change the default user password" sort of platitudes. There has also been nothing available on de-provisioning BMCs securely, from the vendors or any other literature. These would at least raise the bar but can't address the core, more intractable, issues.

It seems that the server vendors are in the driver's seat to really implement change. They simply have to release more details of their IPMI based products; but we server and business owners really need some transparency from them to help with this situation. Opening up access to 3rd parties and customers to permit defenses, logging, and introspection should be de rigor. Telling customers how to de-provision servers in a safe and secure manner should be fully documented. And with so much open source used to implement the BMC more details and released code should be easily found and widely available.

It might also be high time to create an updated IPMI specification; by all means keep the good parts, but backwards compatibility with the worse bits would be simply dangerous.

Researchers, large vendors, and the security community are full of brilliant folks, however, and if I'm sure they'll weigh in on this situation sooner or later. Various vendors have done a thing or three to try and address some of the security issues I've

highlighted, but not much that attempts to address the root problems. And even the nimblest of large companies can be plodding; when I worked at Sun we had a horrific security problem that somehow survived for years ("/etc/hosts.equiv", for those who remember) that by default allowed remote login access without a password. It can (and did!) take a very, very long time to fix even the simplest of problems for fear of losing sales – after all, security is generally pretty low on the sales totem pole.

The first version of the paper met with a small bit of response, but I've yet to see any disagreements about what I wrote other from people who hadn't read what I wrote. Six months ago I brought my issues to the vendors, CERT, the IPMI community; while everyone was polite it didn't seem to accomplish a lot[55].

I've done a lot of work on IPMI since version one, and I'm more confident than ever that the situation is much worse. As predicted, a muted response. I've heard essentially nothing from the IPMI community. Vendors have ignored it as well, although they're starting to patch some of the purely bug-related problems.

Ah well, c'est la vie – I wish us all the best of luck, it'll sort itself out one way or another.

– dan farmer, Seattle, August 16th, 2013

Bibliography

An updated or live version of the bibliography may be found here. I've also some additional details and pointers on my main IPMI page, which should also have the latest and greatest version of this paper. I'm refraining from vendor-specific items unless they're of particular note.

I'd like to thank....

DARPA's Fast Track Program helped me do some initial research on IPMI, funding 4-5 months of invaluable research time on the subject. A big thanks to all on that program (and especially Peiter Zatko, aka Mudge.)

I got invaluable help from a few individuals as well – thank you! Alphabetically:

- **Hank Bruning** – President of Jblade. Hank had some great commentary on the piece as well as having some real numbers and utilization data.
- **Jarrod B Johnson** – Raleigh/IBM@IBMUS. For telling me about the dangers of Cipher Zero as well as passwords being stored as plaintext on the BMC; he has near encyclopedic knowledge of the IPMI specification along with various security implications.
- **HD Moore** (founder of the Metasploit project and Chief Research Officer for Rapid 7) – for his invaluable help on networking and technical stuff, as well as helping me both test out and sharing some code.
- **Jesse Robbins** – co-founder of Opscode. The first person who agreed with me on the dangers of IPMI ;) Great stories of IPMI usage in the "real world" and invaluable sanity checking.

- **Avi Geiger** – For his most excellent knowledge of low-level hardware, wiring my HP for sniffing, cooking up the best chicken soup I've ever had, being a great brew meister and better friend.

Intel gets an honorable mention for being so open to discuss the issues; I can't mention names, but thanks anyway, you know who you were and were a big help. Dell had the kindness to talk to me as well and discuss some interesting issues.

And finally someone who worked in the BMC/IPMI industry engaged me in a series of illuminating critiques and conversations over email. They asked to remain anonymous, but thanks anyway!

---

[1] At least among mainstream operating systems. Any computer hooked up to the Internet would do well to avoid this.
[2] http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-adopters-list.html
[3] Page 15 of the ATEN user manual for their IP9001 pcIPcard.
[4] Version 1.0 had no networking capabilities and is rarely seen these days for obvious reasons.
[5] AMT is dangerous but I don't personally view it to be as threatening as IPMI. There are lots of reasons why I think this, but this paper is already long enough to go into that!
[6] IPMI creates sessions tunneled over the UDP protocol, which is a connectionless protocol.
[7] Cleartext is a phrase that means data sent is communicated in ways that are understandable by anyone watching. If my password were "Josie999", then anyone that had access to the network data would see exactly that string of characters. Encryption is generally used to grant confidentiality for passwords and other data.
[8] I'll use root a lot in this paper, which is the name of the UNIX/Linux all-powerful administrator account. I could say administrator or whatever the system account is on your OS of choice, but for many root is a short-hand way to refer to the administrative account or having administrative privileges. IPMI doesn't care what OS the host uses, which is part of its charm.
[9] The free toolkits seem to have adopted Cipher 3 (arguably the strongest) as a default choice if a 2.0 connection is desired and the user doesn't explicitly choose a cipher, but there's no guarantee that the remote BMC supports it.
[10] Most of the problems with MD5 don't necessarily completely mar HMAC protection, but MD5 seems flawed enough to avoid if a better alternative is available, in my opinion.
[11] http://csrc.nist.gov/groups/ST/hash/policy.html
[12] I've heard rumors of a forthcoming patch, but it's still true at the time of this writing.
[13] A recent beta version of the Hashcat password guessing engine may be used to guess RAKP hashes and has been clocked at about 20 million password guesses on a single Intel core, making this a serious threat to any passwords.
[14] The only reference I could find of someone using it was with Cacote & Masi's "Using the Intelligent Platform Management Interface (IPMI) at the LHC GRID"; they describe machinery to create new passwords every day for every server, but it was only used on about 2,000 managed hosts, and it seems to involve some very complicated machinery.
[15] For system geeks this is a fascinating topic in its own right. An SMI interrupt is a non-maskable interrupt that has the highest priority of anything on the computer. Upon receiving an SMI you're dropped into SMM mode and the normally executing OS, kernel, and application code freezes. Code can be executed depending on the context of the interrupt, and once done SMM is exited and things go back to normal (this might last a few milliseconds, so its mostly invisible to end users.) To people at higher levels things can seemingly happen between CPU ticks – sort of a Matrix-like bullet time for servers. The IPMI spec says it all – it grants "full access to system memory and I/O space". See the bibliography for more details.
[16] Page 24 of the IPMI version 2.0 specification.

[17] In the patent application "System ROM with an embedded disk image".
[18] "HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity"; A.M. Azab et al. Unfortunately the exact method used to generate SMIs from the BMC was received under an NDA from IBM (private communication with A.M. Azab.)
[19] The Linux kernel paired with the Busybox utilities (a binary that emulates considerable numbers of UNIX/Linux utilities in a stripped down and optimized fashion) are very popular in embedded systems in general and on the BMCs I've looked at. I hear that other small OS's are also used on the BMC, but have yet to personally see one.
[20] I've a small list of BMC manufacturers along with the location of their HQ and where they are physically made.
[21] A boot loader is simply a little program that allows a computer to get started. The cleverly named Das U-Boot is very commonly seen in embedded systems.
[22] It can be difficult finding out where the vendors stash their GPL code (if they do at all), but I've listed a few on my site.
[23] Like Chuck Norris, the BMC doesn't sleep – it waits. It still handles its internal matters, deals with low-level system business, and you can talk to it anytime you want.
[24] The net is full of stories about how various vendor implementations of IPMI hopping from Ethernet adaptor to another, and being uncertain which one it actually is listening to even if you try to force it via configuration. IPMI implementations seem to really want to talk to the network, and since it usually starts up before the OS does at times it tries to grab the first network port it senses to be on. This can lead to additional attacker opportunities.
[25] On my Supermicro server the passwords were stored in a file in the file system in plain sight - they're kept in "/conf/PMConfig.dat"; your own mileage may vary.
[26] UDP port 664 is at times used for ASF 2.0 Secure RMCP.
[27] http://www.gnu.org/software/freeipmi/manpages/man8/rmcpping.8.html
[28] When dealing with multiple interfaces Ethernet bonding is used to set up the slaves and permissions. I missed classes on bonding and embedded systems while growing up, apparently, so I'm not very conversant on this
[29]For instance, to mount a remote CIFS – "**mount -t cifs //10.0.0.1/media /tmp/mnt -o user=root,pass=toor".** I only put this here since there's no discussion of it anywhere that I can find.
[30] Apparently some Linux distros now have autorun also, traditionally a Windows feature that automatically executes code when a CD or other removable media is inserted. It might be interesting to try virtually mounting USB HID images to attack the keyboard.
[31] Forensic Data Recovery from Flash Memory, M Breeuwsma et al. appears to among the most highly cited works on the topic, and a good read about how to recover such data. There are many papers, tools, and references out there.
[32] Perhaps someone could figure out a way. I was not able to, but I'm also not an expert at this stuff. Certainly an implementer could figure out something. Maybe we could ask the Chinese to let us know their backdoors so we could get in…. I kid, I kid! :)
[33] More about this at http://fish2.com/ipmi/dell/secret.html
[34] A special case of this is if you are logged in with administrative privileges on a server – in this case no authentication is required to execute IPMI commands on that box.
[35] Monitoring is possible, of course, but typically done sporadically and for trouble-shooting and performance reasons, not typically for security.
[36] Perhaps there is some large enterprise that can make it work, but the larger you are the harder it gets.
[37] The IPMI specification has something they dub the "Firmware Firewall", but it doesn't have anything to do with networking; it's an attempt at blocking configuration, messaging and write operations on interfaces, and appears to be mostly designed for use in big blade servers. It also seems pretty worthless (it's complex, no examples exist in the world, and no software to test or audit the configuration, etc.), but perhaps it's used somewhere.
[38] This is an unfortunate by product of the IPMI specification, which sometimes says you have to use or send the password in unencrypted form. Some vendors put up a fight here, and try to block

sections of the ROM from casual reading, or semi-encrypt the passwords and extract them at runtime, but an attacker can still capturing the BMC's RAM or reverse engineering the boot process to reveal them. Admittedly with limited experience, but it seems simple to access the IPMI passwords once on the BMC.

[39] See my web site for an enumeration of at least some of the methods. And compromising a server by physically attacking it is only cheating if it only grants you access to that single computer, not thousands of others.

[40] I'm actually not facetious here – there are many times in the protocol where the user experience is cited as a reason for certain designs, which I thought was wonderful, as you almost never see this. Unfortunately, they went a bit overboard at times.

[41] The BMC has the potential to read and write directly to them as well.

[42] It appears that some vendors try to put a hard-coded wall between the server and the BMC to disallow server-to-BMC network communications. If this can't be circumvented, an attacker may simply talk to the BMC from a 2nd accomplice computer after changing the BMCs network address.

[43] These hyper-crucial, crown-jewelesque, super-back end types of assets also have some of the worst security in your environment. I wrote an essay about this that may be found at http://trouble.org/?p=262.

[44] Unlike regular hard drives flash memory can only be written to a fairly small amount of times before failing. It might be possible to kill off a BMC with a lot of brute force writing, but with the implementation details so scarce I wouldn't count on it to be a reliable way. A small nail and a ballpeen hammer might be effective at killing the BMC, but who knows if that'd kill off any server functionally – the Southbridge and BMC have an odd relationship and it might have undesired consequences. Some vendors may have a setting to really disable the BMC.

[45] This is in part because of the intertwined nature of IPMI and the server combined with the brittle nature of the BMC architecture and security model. Root accounts on a server can create local administrative IPMI accounts without any additional authentication. The most straightforward way is to simply reboot the system onto media of your choice and mount the local drives; you may then install a new account, a new OS, or do whatever you wish – after all, provisioning servers is one of the basic uses of IPMI.

[46] This is definitely true if you have compromised the BMC, and possibly true if you simply have IPMI control, depending on your server vendor as well as how you run your network and manage IPMI. If you use a dedicated, IPMI-only management ethernet jack on your server I don't know of a way for a server to communicate with that interface. But if you share physical connections or use a network interface that the server can access then this is true in general as well.

[47] Pretty much everyone can run our old friend telnet along with SSH, VNC, web (http/https), email, the IPMI protocol itself as well as a variety of custom programs to serve remote virtual media along with other IPMI custom services. They are almost always daemons or agents are running on the BMC.

[48] It's mandatory to keep embedded systems rock solid and stable because of the difficulties of patching and maintenance, so vendors don't want to mess with anything once they reach a stable state unless they deem the problem critical. Security issues, if patched at all, will usually be rolled out in the next normal maintenance release; the patch notes won't usually mention security but will focus on features or perhaps mention stability or some other veiled phrase.

[49] There are more details about this in section 3. I assume that this is because vendor probably realized that you don't want people reading those passwords, and attempt to block you from reading the flash storage or mucking with the IPMI subsystem. This also means that you can't make backups of your flash or tell if it's been modified, as a compromised BMC can simply say what you're expecting or wanting to hear.

[50] At least as it stands today; this will presumably change over time as people gain more exposure to IPMI issues, tools and procedures get developed, vendors change, etc. Even if any forensic tools or methodologies existed for BMCs you couldn't use them unless you could login to the BMC to kick the virtual tires.

[51] When communicating to BMCs software as software goes from IPMI version 2.0 (sometimes encrypted) to version 1.5 (never encrypted) or when using various vendor services you'll get

encryption sometimes. It's usually hidden, undocumented, or you have to hunt to figure it out (or drag out the packet sniffer.)

[52] This is a presumption on my part, based on the evidence I've gathered so far. At least some of the data may be sniffed; but in an out of the box install not all servers allow network monitoring from the server to the BMC or vice-versa. Data is admittedly in seriously short supply for me here. However, I have been able to watch the BMC from a Dell server, and it seems a foregone conclusion that given the BMC's power listening to the server's network traffic would be fairly straightforward. It could be a matter of changing network or kernel settings, the network card's firmware or configuration, or perhaps even that some vendors have figured a way to really separate the two. I've written a bit more about this on my web site.

[53] A zero day exploit is when an attack exploits a previous unfixed or unknown security problem that leads to a system compromise. Once it is finally reported to the vendor (or, conversely, to the larger public) the clock can start ticking to fix or address the problem. Bruce Schneier wrote a nice piece in Forbes about the economics and ethics of zero day exploits.

[54] One of their write-ups, a guide for penetration testers, covers some of the more pressing main technical issues.

[55] I saw that CERT just put out an advisory on this; they didn't want to when I first told them, I guess someone convinced them.