Version 1.0

dan farmer
zen@trouble.org
1/28/2013

IPMI: Freight Train to Hell

Abstract

Intel's Intelligent Platform Management Interface (**IPMI**), which is implemented and added onto by all server vendors, grant system administrators with a means to manage their hardware in an Out of Band (OOB) or Lights Out Management (LOM) fashion. However there are a series of design, utilization, and vendor issues that cause complex, pervasive, and serious security infrastructure problems.

The BMC is an embedded computer on the motherboard that implements IPMI; it enjoys an asymmetrical relationship with its host, with the BMC able to gain full control of memory and I/O, while the server is both blind and impotent against the BMC. Compromised servers have full access to the private IPMI network

The BMC uses reusable passwords that are infrequently changed, widely shared among servers, and stored in clear text in its storage. The passwords may be disclosed with an attack on the server, over the network network against the BMC, or with a physical attack against the motherboard (including after the server has been decommissioned.)

IT's reliance on IPMI to reduce costs, the near-complete lack of research, 3rd party products, or vendor documentation on IPMI and the BMC security, and the permanent nature of the BMC on the motherboard make it currently very difficult to defend, fix or remediate against these issues.

Introduction

Imagine trying to secure an important group of servers that are essentially black boxes: not only are they full of old code and attack points, but there isn't any documentation on how they operate; you're unable to login, patch or fix problems; you're unable to run any server-based defensive, anti-malware, or audit software; and to top it off all the servers share reusable passwords that are stored in clear text. These insecure servers are not only pervasive but allow stealth control and monitoring over all the other servers on your network. Welcome to the 21st millennium.

There are at least twice as many servers in your data center and networks than you might think. Modern servers have an embedded computer called the Baseboard Management Controller (BMC), which has its own CPU, RAM, storage, and physical network interface that all operate independently of the main server. The BMC was designed to facilitate out of band (OOB) operations and implement the Intelligent Platform Management Interface (IPMI), a standard created by Intel and a consortium of large server vendors; today nearly 200 computer manufacturers are on Intel's Adopter's List[1].

IPMI is mostly used for low-level tasks like rebooting servers, monitoring physical sensors (e.g. temperature, memory health, fan speed, etc.), providing virtual remote consoles and so on. However, due to its design, IPMI could also provide a mechanism to spy, control, and modify data and network traffic in a fashion that is about as close to invisible as can be imagined. To top it off IPMI access is governed by a clear text (i.e. unencrypted) set of passwords stored on the motherboard between all servers in a management group.

That thanks to IPMI and a widespread confluence of issues – including vendor design and architectural decisions, the standards and resulting implications on operations and security, how it is used operationally within organizations, and an almost complete lack of awareness or dialogue or research on the subject –we've dug ourselves a serious digital hole about as large as the Grand Canyon. To compound matters it's not a single problem – none of them individually are showstoppers – but due to the cascading nature of issues the problems add up to something more. Unfortunately all these factors take a bit of explanation, hence the length of this document (I've also written a Reader's Digest version, but it might give more questions than answers depending on your knowledge level.)

I must note up front that this paper's title is a bit unfair to IPMI, which does have its own issues, but those are greatly amplified by factors that have nothing to do with the specification. But in an attempt at clarity and simplicity I'll be using "IPMI" as a catchall phrase for several things – the specification, the BMC, how customers actually utilize it, the vendor and how they add features on top of the spec, etc. I hope the IPMI experts will forgive me for potentially slandering pure IPMI, and I'll try to distinguish who is the root cause of the various issues.

The severity of the situation mainly depends on three factors: the amount of servers in IPMI management groups, the amount of time between password changes within groups, and the difficulty of exploiting or abusing the BMC or the IPMI interface. If I'm off the mark on any of these it could be much less damaging than I claim. This is further clouded because hard data is perniciously difficult to get on such sensitive subject matter. In any case this paper details my analysis to date.

Those that are familiar with IPMI can safely skip or skim section I. I cover some architecture and implementation details of the BMC in section II; IPMI usage and deployment in the wild in III; section IV has a list of specific security problems, and then I end with a summary and some thoughts on where we might go from here.

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 out of the path of general reading. And while I thank some 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, so 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.

Disclaimer

I'd like to emphasize that I'm not an IPMI expert. I'd never used IPMI, and knew about as much as most people (e.g. nothing) until a couple of months ago when I got curious. I'm doing this completely on my own free time, although I did leverage three servers that are for my DARPA based Cyber Fast Track program research.[2] This handful of servers along with some network scanning, a variety of conversations with knowledgeable people, and a whole lot of reading is what I base the entire paper on.

I did contact CERT and some vendors about the issues in this paper but no one has been interested in any sort of formal response or communication. Indeed, there is a definite sense that the situation isn't as dire as I proclaim. However, I'll note that this all has been implemented and

expanded by vendors with absolutely no scrutiny or oversight from the security community and researchers, which hasn't traditionally been a metric for security success. Finally: this isn't meant to be a scientific paper with measurements and hypothesis and results; given the breadth of the topic it's more of a porky watercolor, a sketch, or essay that attempts to draw fairly broad conclusions on a bit of disconcerting evidence. I'll stand by my work and am content to disagree with the world at large (as is my wont), but will remain open to discourse if anything comes of this, and will be happy to change my mind as I learn more.

Table of Contents

I. IPMI

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.)

The IPMI system is actually composed of three specifications: the "Intelligent Platform Management Interface (IPMI), Intelligent Platform Management Bus (IPMB) and 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 is still in fairly wide use, came out in 2001, and 2.0 was rolled out in 2004 (the last revisions being published June 2009.) 2.0's main security contribution was to introduce the option of network encryption.

Server and firmware vendors add features and use a variety of names; Dell calls theirs iDRAC, Hewlett Packard iLO, IBM IMM, etc, but in the end it's all IPMI under the hood. Intel launched a similar effort for personal computers called Active Management Technology (AMT) that shares many features with IPMI, but while hazardous I don't personally view it to be as threatening as IPMI.

IPMI was designed to be resilient – the BMC is able to communicate with its server or with other computers even when the network is 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.)

The BMC essentially provides a data abstraction layer to the server's physical hardware via an IPMI programming interface. This makes it simple for monitoring tools such as Nagios, Cacti, etc. to extract this information on an ongoing basis via IPMI rather than mucking around with the firmware or sensors directly.

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. IPMI can also reboot the system upgrading the BIOS, install an operating system, or other low-level management tasks.

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.

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.

Commonly the BMC is physically connected to the South Bridge, a core component on a motherboard that controls most I/O. 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, I^{2}C etc.

Figure 1, courtesy of the Intel Development Forum, might help illustrate IPMI's place in the management stack – traditional management systems deal with the higher level concepts of applications, operating systems, and the like – and something was needed to deal with the low-level hardware issues (IPMI is also used by supercomputers, virtual controllers, clustered computers, etc. to spin up and down new resources on demand.) 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. Support for popular scripting languages and web interfaces are de rigueur.

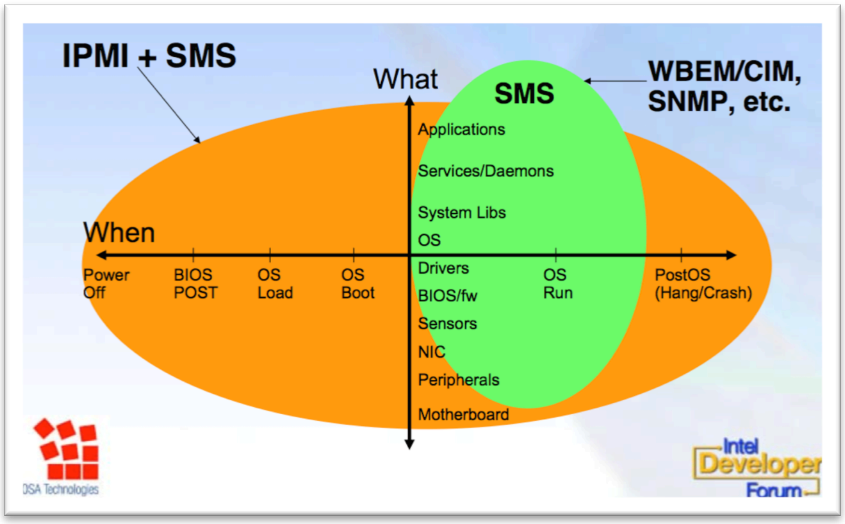

Figure 1

From the start OOB was meant to communicate when the bits hit the fan; initially people skipped the normal user to server communication channels and used such reliable but slow or pricey technology like analogue POTS lines, dedicated Internet connections or other independent networking technologies; the important part was that it was separate from the main network to be protected from the same outages that might bring down the network. This also gave an important amount of security separation, making it very difficult for anyone but insiders to know anything about the OOB machinery for a given network or datacenter. As time went on, interaction was deemed more desirable (e.g. video, mouse, etc.), cost cutting became more important, and the communication separation became less frequent; these days instead of having a network completely devoted to OOB its more common to segregate traffic with VLANs or share a non-routable network with 3rd tier database or other non-internet facing servers.

If you're only worried about the one-password for all those machines, the IPMI specification itself does offer some help of sorts in the RMCP+ Authenticated Key-Exchange Protocol, or RAKP. Cacote & Masi describe[4] using this to create new passwords every day for every system, but it was only used on about 2,000 managed hosts, and it seems to involve some very complicated machinery. RAKP has flaws as well, however, including allowing people to use null passwords (making it useless), along with the fact that the IPMI specification says that it doesn't have "a secure, confidential mechanism for installing and distributing user keys between BMCs and remote consoles", which dampens my enthusiasm. Search engines show very little documentation or examples of use, which probably means not a lot of people use 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, and yet there's almost no community, research, or help for users wanting to secure their systems. There's a real need for security checklists, articles, a security FAQ, software tools, and discussion about the implications of IPMI and all it entails.

I've placed some additional technical security notes as well as pointers to other resources on my site.

II. The BMC

There is a wealth of information out there about the functionality of IPMI, but it's hard to find much at all about the BMC or the various vendor implementations. It's a real server with a real OS and some fairly complex interactions with its host server and the outside world. So this is a high level gloss is primarily based on explorations with my lab machines along with BMC flash upgrades from other vendors. If you're keenly interested you might read a slightly expanded version[5] I wrote with some additional details.

Figure 2 shows a high-level hardware architectural diagram of a fairly typical BMC, in this case a Winbond WPCM450:

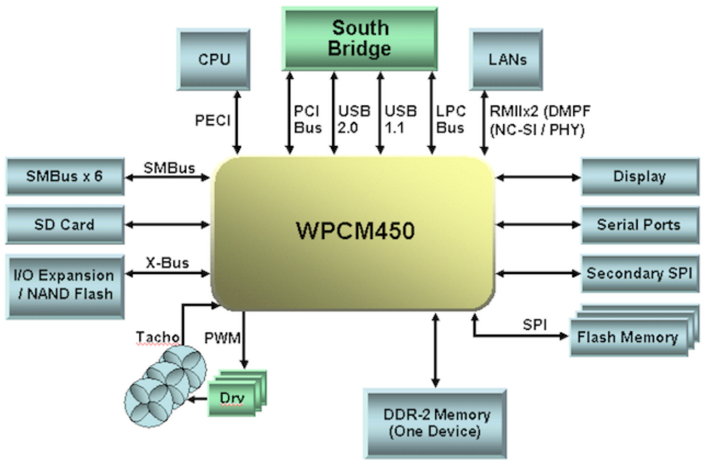

Figure 2

Obviously implementations will vary, but the BMC is hooked up to the Southbridge, the area on a motherboard and responsible for the server's I/O. The BMC is a small computer running a minimalistic OS (Linux being common[6]) 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 major subsystem vendors that design and manufacture the IPMI ecosystem and add-on firmware for the major server vendors, seemingly all made in China[7].

The ARM 926EJ's datasheet, used by Nuvoton, the manufacturer of one of my BMCs, said it cruises along at about 200 MFLOPS – faster than the first Cray super computer. Since the BMC actually does anything useful about 1% of 1% 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 – the kernel, OS, boot loader[8], most of the network services, etc., so in theory the vendors should be sharing the details of their implementations; unfortunately I've not been very successful at finding many details or how to pry such things loose.

The BMC isn't a mere parasite: it runs independently of the main operating system and is always running as long as any power is supplied to the computer. Other than the standard IPMI interfaces used to query its data I know of no software method that would discover activity or details within a BMC unless you're logged into the BMC; physical monitoring via JTAG or other instrumentation might prove fruitful. All communication is handled by the BMC's kernel and supporting programs, so even the most elemental of requests could return false data.

The IPMI specification explicitly mentions how useful System Management Interrupts (SMIs) can be for IPMI, although they're not officially part of the specification. SMI's are the highest priority non-maskable interrupts on a computer, and when asserted the processors are shifted into System Management Mode (SMM[9]) 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. At this point the IPMI specification puts it succinctly – the BMC has "full access to system memory and I/O space[10]."

After scanning lots of hardware literature, marketing material, and vendor documentation it seems as though the BMC usually seems to have the ability to use SMI handlers to enter SMM mode. Even if it didn't, however, with its low-level hardware connections and existing power over its host it would be hard to stop it from doing anything it wants.

Communications

There are four main ways of communicating to the BMC directly: there's an interactive shell, a web interface, various command line tools (like *ipmitools*; by default this is on UDP port 623), 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.

Most of the time IPMI must be explicitly turned on[11] via the BIOS/UEFI/system firmware or as part of a special vendor custom order (there are stories of some vendors having it turned on by default, which is a serious risk), 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 management port for IPMI, but it can also share the network adaptor of the host OS[12]. Once a power cord is plugged in the BMC will start up, whether or not the host system has been activated.

Authentication

IPMI authentication is handled via a small set (usually from 10-16) of local IPMI users can be created on each channel of the BMC. They can be given passwords of up to 16 (IPMI 1.5) or 20 (IPMI 2.0) characters, which are both stored in clear text[13]. Because the client never sends the password (cryptographic hashes are used instead; see the end of the previous section) IPMI also can't use out-of-the-box standard network authentication; while most vendors claim to support the usual suspects like LDAP, AD, RADIUS, etc. they have to resort to some sort of hackery (there are rumors that in some cases they simply append the plain text password as an attribute, not quite what you might want; in any case it's different than what one might expect.) But even if network authentication is used you still want that native, host-based fallback mechanism so you'll have console access during emergencies – for instance when the network or RADIUS or AD servers are down; indeed some vendors don't let you disable the BMC based authentication at all to prevent you from locking yourself out.

Most IPMI actions explicitly require a password to be entered. There are a few special cases, however.

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) is enabled any user may be logged into without any password. It's usually possible to store SSH certificates on the BMC to allow a password free login (with the appropriate private key, of course.) This may be problematic from a management standpoint; who knows if that SSH key is valid, or whom it's really from? You 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. Some vendors just stick it in a file[14] in plain sight, but it appears that most try to at least nominally hide it.

Network Services

A BMC generally runs a half-dozen or so network services out-of-the-box, but a dozen or more are typically available through the web or command line interfaces. Among the usual cast of characters there's web (HTTP and HTTPS), SSH, telnet, SMTP Virtual KVM/Keyboard/Mouse, Network USB and/or Virtual Media, SOL (Serial over Lan), 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.

Of particular and perhaps visceral notice is that most BMCs now offer hooks to Microsoft's RDP/Terminal Services and 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.)

Virtual Media/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 don't think any vendor doesn't offer the feature on the BMC, although sometimes only with their advanced or enterprise version (for an additional fee, of course.)

Using the virtual media feature you 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 thje host exactly like their physical counterparts (beware of autorun![15]) Virtual media is heavily used for provisioning or bootstrapping new servers or applications, remotely installing or reinstalling the OS, deploying diagnostic tools, etc.

A few security notes

I won't go too far into the security possibilities here (see section IV for more on that) but 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. The BMC is constructed with many millions of lines of code; a mixture of moderately popular open source blended with propriety code from a small number of vendors. I haven't found any non-vendor studies or efforts to audit or test for security issues. This is what they call in the biz a green field opportunity, and there is a virtual certainty there are many, many vulnerabilities yet to be uncovered. A few areas ripe for exploitation:

- The web server –BMCs have a large number of web pages that execute JavaScript, accept user data and input, transfer data over the network, etc.
- More traditional but non-web network services (for both authenticated and non-authenticated users)
- Network stack (TCP/UDP/VLAN/etc.)
- IPMI protocol handling
- Logging & event handling
- SMASH/CLP interactive shell
- Server to BMC communications – IPMI protocol but also the rest of the bus and low-level communication the kernel has access to
- Cryptographic attacks (certificates, protocol handling, etc.)
- The surface or interface between low-level hardware and BMC management and communication software

Recent research suggests that older code is more vulnerable than recent; because of the nature of embedded systems and the frequency of firmware updates the code on BMCs will be usually be several years old, adding to potential danger.[16/17] There has also been a lot of research and work done with embedded Linux on routers and cheap Wi-Fi boxes that could presumably be leveraged to investigate BMCs.

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."[18]

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. An ephemeral OS booted in RAM could explore the disks, applications and data, which could be folded, spindled, and/or mutilated. This is fast, as well; this could take only a minute or two, 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.

While you can turn off some of the network services, others often can't be shut down (like the web server that allows you to configure the system) unless you turn off IPMI from the BIOS configuration. And even if a service is disabled they can easily be re-enabled if an attacker has physical access to the server, an administrative IPMI account or root on the server.

Vulnerabilities uncovered are presumably shared for all servers sharing the BMC firmware for a particular vendor – and, if the firmware manufacturer writes code for more than one server vendor, they may well be shared by other BMCs as well. Unfortunately it's not widely known who the underlying maker is of a particular BMC is.

And these are special vulnerabilities: they don't grant direct access to the **server**, they instead give you **shell on the BMC**, which is a very powerful place to be.

This is an unusual type of vulnerability: normally when a problem is found we think it's not a big deal, we can just remove the software or apply a patch. In the case of IPMI you can do neither. It's really a pernicious problem –if a vulnerability exits it means that you have latent backdoors into your servers that can be enabled at any time either by the BMC or the server – an attacker could easily set up scripts that (re)enable the problem at regularly timed intervals or in other nefarious ways.

And who knows what 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[19] that would enable shell access on the BMC one might wonder: what else is out there on these black boxes? 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.

Finally, a gut feeling – although I've admittedly a small sample space, in the handful of BMCs I've examined it all seemed... well, a bit sloppy. There are developer files left lying around (like "1ogin.html" – starting with the numeral 1 – and another named "login.html" in the main web content area), file duplications in different areas, poor coding practices in the web pages, system startup and maintenance scripts[20], many binaries that the BMCs didn't need (Dell had such gems as *tcpdump* and *gdb*[21]!), 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), etc., etc. I would also get different behavior if using the Web or directly talking to the IPMI interface also with respect to login timeouts and other issues. Combine this with the crashing and flakiness; when I've seen such messiness it spoke of seeming poor process and execution, and, more to the point, an indicator of security problems that are simply waiting to be unveiled.

III. IPMI in the Wild

Other than the people who actually use IPMI almost no one seems to know what it 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 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 (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 rare, but it still happens from time to time. 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 authenticate oneself[22]; since network authentication isn't commonly used a single password (or a small set of passwords) is shared for large blocks of computers. **The IPMI networks 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. This is an important distinction, and I'll repeat – IPMI passwords are generally duplicated throughout blocks of servers that are aligned with 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[23].

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[24].

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.

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.

IV. Security Proclamations

Here's where I go from background facts to my own IPMI security assertions.

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.

I'm going to try and focus on systemic security issues, not any vendor bugs or implementation details. Unfortunately the BMC server is lacking some of the very basic security controls we now take for granted in modern server – 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 that if you can audit and verify, if not enforce, you may have a chance of getting some security, and if you can't you're out of luck. For starters there are no means of auditing the BMC or performing internal configuration, integrity, and systems detection or change management; but the list is almost inexhaustible: there's no activity logging available, it's impossible to install 3rd party security or defensive software on the BMC, no host-based firewall[25], no method of enforcing or checking password complexity & strength, default there are administrative accounts with well-known or no passwords at all that are hard to check for, there is no documentation, no means of backing up the system, and on and on.

In addition to the basics there are some additional – and in some cases rather unique – problems that make these especially serious:

a) The reusable passwords used for IPMI authentication are saved in **clear text** in flash memory[26]. 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[27].

b) 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.

c) If you can get any 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.[28]

d) What you can do as an administrator or **root**[29] 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

e) 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. IPMI was specifically designed to manage your server's hardware, and can disable, enable, or muck with just about anything on it. Your servers are the world's oysters now. Or… something like that.

f) If a server or the IPMI 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[30]. 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.

g) 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)[31]. Not so much.

h) 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[32]. All the programs are fairly old[33] 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.

i) 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.

j) 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.[34])

k) The BMC can monitor the host computer, but the host server has almost no visibility to what's going on in the BMC[35]. 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 ask a potentially compromised BMC to answer truthfully.

l) It's hard enough to tell if a server has been compromised. It's nearly impossible to tell if the BMC is[36]. 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! Turtles all the way down.

m) Virtual systems can be chewed up from the inside. Of course this isn't a great surprise given all the rest, but there's a paper[37] that explicitly describes using the BMC on an IBM server to generate SMI interrupts to 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.

n) 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[38].

o) 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.

p) De-provisioning systems really sucks 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? Do hashes remain or will the passwords remain burned into the flash storage, or…? 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 anything other than physically destroying the motherboard should be trusted.

q) 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.[39]

r) The 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[40].

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. They now have apps that store the IPMI password, and you know how often phones get lost, stolen, or left alone and defenseless in a hotel room….

All of this makes **known** incidents very troubling as well. What is the correct response if you suspect that a server has been compromised? And 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. While the computer forensics field is starting to get some maturity, there is a complete lack or research, tools, or technology on BMC forensics (although general embedded systems security research, led by smartphone and mobile device research, has been heating up lately.)

The ability to cross server boundaries and access both IPMI as well as the server network illustrates a shortcoming of the current models of network architecture. Today you have servers on various networks and zones whose connectivity is managed with firewalls, load balancers, switches, and the like. If you have a back channel used for OOB communication an attacker could sidestep the main network and security architecture. That big flat network, full of back doors into all your the servers in the entire IPMI group, plus access to the super-valuable areas you really don't want people mucking around with[41], could now be open for business. The usual split between the logical network topology based on applications and business units is only true on half of the equation, now you have additional overlapping zone of trust based on IPMI groupings.

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[42] 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.

Honestly, I fully expect – without any proof but from some smoldering guns in source code and boot images that I've scoured – that the various vendor sales engineers, developers, and the like have secret methods to access the shell on the BMC for emergencies, trouble-shooting, debugging, and the like.

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 server when you can't login to it, patch or fix problems, run defensive, anti-malware, or audit software on it; your basic black box with no documentation or information available. It shares passwords with a bunch of other important servers, stores them in clear text for attackers to access, has almost no logs and lots of attack points, and you have to run it or take a big financial hit and throw away remote OOB operations altogether, and even that might not work since you can't always turn it off. While I'm not a big fan of the "weakest link" security theory, in IPMI this strikes me as being a fairly apt statement; your goose might already be cooked and you're simply asking for the orange sauce.

V. Into You Like a Train (Conclusion)

I mentioned at the start the severity of the situation seems to depend on 3 main factors: the amount of servers in a given IPMI group, the length of time between password changes within that group, and the difficulty of either compromising a host server or BMC within that group. If reality is "small", "not-to-long", and "hard", respectively, the problem is smaller than the reverse. Obviously I don't think it is, and in truth, having brought the issue to CERT, some vendors, and a variety of others who don't seem too concerned, I'd freely say that for now I'm in the minority in my belief.

And while I may have painted a bleak scenario, but the confluence of issues in the IPMI specs, 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.

You might ask if I really believe all of this (irrespective of whether or not it turns out to be true) then why in the heck did write this all up, am I intentionally trying to cause problems? I'm actually trying to help – talking to people about this over the last few months convinced me that no one was worried, but more than that no one knows anything about IPMI et al except a rarified few. Surely I'm not the only one to come up with this line of thinking, but there is almost no discussion or research out there on the topic. Indeed, it would be rather surprising if it were not being currently exploited under our virtual noses. And any discussing any of the individual issues doesn't allow one to connect the dots; it's the little bits that add up to something greater, not the little bits themselves that are important. I hope some dialogue and change might start, but I've been wrong many more times than I'm right.

On the bright side, 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. On the downside: who would actually know it existed unless it was found? Using a BMC as an attack or spy platform could 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 BMC developer to do well – not simple for most, but attackers with the right skill set shouldn't have much trouble. 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 probably won't help much. RAKP (RMCP+ Authenticated Key-Exchange Protocol, briefly discussed in the IPMI section) is possibly of use, but key distribution and the simple pain of management is not insignificant. Ensuring that passwords are better protected and not so widely shared along with mandating network encryption would certainly help, but since IPMI lives at such a low level it's very difficult to protect yourself against if the computer can stab you in the back when you least expect it. TPM or other trusted boot mechanisms aren't commonly deployed (and especially not for servers) but most designs 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. Presumably some people have come up with effective solutions but I've been unable to unearth them. Unfortunately firing up a large management system always takes a lot of effort to start, and even if successful they themselves become a very large and valuable target.

De-provisioning of servers should be a more serious concern now; if you don't destroy your BMC now when a server ends its service you might wish to revisit your process and policies about how to do so in a safe manner.

We do seem to be a bit behind the ball right now – there doesn't even exist a reasonable set of technical security and configuration best practices, which is pretty amazing. 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. Also there is absolutely 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; transparency seems nearly a requirement 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 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; perhaps they've already figured all this out and are simply waiting for their chance to spring, or I'm not in the "in" club who hears about such things. 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.

However, if there is any response to this paper I might anticipate the IPMI experts to say that IPMI isn't the problem – it's instead what the vendors did layering on all that additional functionality on top of the specification; the vendors might balk and claim that none of this applies to the very latest version of their particular implementation (if at all), and if customers would only listen to them and simply did what they were told to do all would be well. And the customers would be in the same place as before. These are valid critiques that don't change any of my conclusions, but I'm open to learn. C'est la vie, and best of luck.

– dan farmer, Seattle, 1/28/13

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've put a few at the section at the end.

General IPMI

Start with the source; Intel has put out a variety of documents; in particular the IPMI specifications, for 1.5 and 2.0 (484 and 644 pages of deathless prose, respectively) and the IPMI CIM Mapping Guideline were invaluable:

· IPMI spex - all versions
· IPMI/CIM Mapping guide

There's a detailed (but not complete) fairly technical write-up on IPMI basics by Corey Minyard in 2006

· IPMI - A Gentle Introduction with OpenIPMI.

It's hard to understand IPMI/BMC land without some knowledge of flash - NOR is a "random access device appropriate for code storage application", while NAND is better for storage (you can't directly execute code from NAND disk -it "must be loaded into RAM memory and executed from there."

· NAND vs. NOR flash - a Flashy writeup

MTD - memory technology device; an abstraction layer for raw flash devices (NAND, NOR, etc.) Some very useful background information on MTD:

· http://www.linux-mtd.infradead.org/

Software

Four very high quality IPMI software packages: freeipmi, ipmitools, ipmiutils, and openipmi; in addition to the software they have some excellent documentation, writeups and details about the world of IPMI. While perhaps not possessing the most imaginative of names They're all worth checking out:

- OpenIPMI
- FreeIPMI
- IPMIutil
- IPMItool

FreeIPMI in particular has amazing documentation and is used in many vendor offerings. Finally, there's a really nice (and fair) comparison of them on sourceforge written by one of the authors.

I used a ton of tools, way too many to mention more than a few. On any sort of unix/linux variant strings is just such a frickin' great tool... use that on any binaries along with "hexdump -C"; strace is also godly.

Binwalk and the firmware-mod-kit were also useful in unraveling some details. Luigi's signsrch provided some color commentary (windows only, but could run via wine; e.g. - i "wine ~/signsrch.exe binary.file".) Qemu was invaluable for emulating some Arm processor things and DosBox saved me from having to dig out the ol' DOS floppies (DOS may never die; assorted low-level system vendor programs still run via DOS.) With the exception of DosBox and the 4 IPMI utilities up there just about every tool had terrible documentation or was difficult to get working on most of the systems I had (Qemu in particular would be even more astonishing it would only run correctly.)

Mac Tools

- Wine was great for running the odd windows executable.
- DosBox - a really impressive DOS emulator for running tools meant for floppyhood or whatever.

(mostly) Linux Tools

- MTD utils. The very useful but so-appallingly-documented-that-you-can't-belive-it-was-written-by- anyone-with-a-desire-to-communicate-with-other-humans MTD utils, Several Linux distros seem to have this as a package; source code avail at git://git.infradead.org/mtd-utils.git.
- lm-sensors. Finding out about the hardware and helping monitoring computers thingee. Hint - just type sensors-detect and follow orders.
- flashrom. Identifying, reading, writing, etc. flash chips. This is a pretty damn cool tool. And another in a long line of so poorly documented utilities you think they don't want you know how to use them tools. It used to be easy - "flashrom -r /tmp/foo" would dump your flash into that file name. Now they require you to specify all kinds of crap on the command line and have pretty much zero examples of how most people might use the tool and a man page that is... well, a man page. They specifically say don't write shell scripts to use the tool because the options will change again. How friendly. If version .94 works on your system you might try that. A really interesting tool nearly destroyed by unbelievably poor documentation for the beginner.

Mucking with firmware, assorted links, papers, etc.

IPMI stuff is all about embedded systems; a really nice intro to such things is Christopher Hallinan's book, which is simply an excellent book, especially for modestly technical beginner's such as myself:

· Embedded Linux Primer: A Practical Real-World Approach
· Project Maux Mk.II (And Mk III as well.) A talk on to install SSH on a NIC card. Arrigo Triulzi arrigo@sevenseas.org (Arrigo's homepage: http://www.alchemistowl.org/arrigo/)

Additional reading

Details on SMI/SMM:
· Wiki page on SMM/SMI
· http://cs.gmu.edu/~tr-admin/papers/GMU-CS-TR-2011-8.pdf , J. Wang, K. Sun, and A. Stavrou, a GMU technical report.

A paper that discusses using IPMI to generate SMIs to enter into SMM mode:

· HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity, by 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.)

CERN used a set of programs to generate daily random IPMI passwords to manage just under 2,000 servers - a nice write-up here:

· Using the Intelligent Platform Management Interface (IPMI) at the LHC GRID, by Hugo J. M. Cacote & M. Masi, 2007.

A nice overview of **AMT** security (the IPMI-like thing in PCs and such by Vassilios Ververis:

· "Security Evaluation of Intel's Active Management Technology".

Joanna Rutkowska on using a USB stick to compromise encryption keys in general is worth reading; she dubbed it:

· the Evil Maid Attack,

There are many references and tools to aide in USB sniffing; here are some Linux references, but typing "USB sniffing" in any search engine will get lots of others.

· Linux USB tools

A paper on forensics & flash storage; from SMALL SCALE DIGITAL DEVICE FORENSICS JOURNAL, VOL. 1, NO. 1, JUNE 2007; Marcel Breeuwsma, Martien de Jongh, Coert Klaver, Ronald van der Knijff and Mark Roeloffs:

· Forensic Data Recovery from Flash Memory

· Lessons Learned from Five Years of Building More Secure Software, M. Howard, 11/2007 MSDN Magazine.
· Milk or Wine: Does Software Security Improve with Age A. Ozment and S. Schecter, 2007 USENIX Security

Vendor stuff

I've downloaded many BMC ROMs and have read through more vendor manuals and than I can count. Here are a few highlights.

Dell's security overview for iDRAC 6:

· Integrated DellTM Remote Access Controller 6 Security

HP's security overview for iLO 3:

· The HP Integrated Lights-Out Security, 7th edition

Darren Cepulis/HP's patent application has some interesting details on using SMIs with virtual disks (and quite possibly sheds some light on how HP implements such things in iLO.)

· "System ROM with an embedded disk image"

I'd like to thank....

I got invaluable help from a few fine folks – 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 excellent explanations of some of the knotty salient IPMI details.
- **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 Ruben** – 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.

[1] http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-adopters-list.html
[2] This paper isn't part of my DARPA work or their fabulous Fast Track program, but does leverage some of the experiences I've had in it. I'm currently writing a simple network scanner to locate IPMI systems and hope to create some best security practices to audit against.
[3] Page 15 of the ATEN user manual for their IP9001 pcIPcard.
[4] "Using the Intelligent Platform Management Interface (IPMI) at the LHC GRID", Hugo J. M. Cacote & M. Masi, 2007

5 At http://fish2.com/ipmi/.
6 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.
7 I've a small list of BMC manufacturers along with the location of their HQ and where they are physically made.
8 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.
9 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.
10 Page 24 of the IPMI version 2.0 specification.
11 I wonder; if IPMI hasn't been enabled, the BMC is presumably running and waiting to hear from the outside world; you should be able to talk to it at all times, irrespective of whether IPMI is on or not, but if you could do so via the network… that'd be crushingly bad.
12 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.
13 I see various vendors claiming to have secured the passwords; I'd surmise that this might be accomplished by using a unique hard-to-read hardware SSL key that would decrypt the IPMI password from rest so it may be used by other programs. Two points – first, it's very difficult to keeping sensitive keys from entering into memory, and secondly you could still capture passwords as legitimate users offered up password to the system via IPMI, the web interface, etc. I haven't had the chance to look at such a system in any case.
14 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.
15 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.
16 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.
17 While 's not conclusive, recent research seems to indicate the older the code the less secure it is. See "Milk or Wine: Does Software Security Improve with Age?" (A. Ozment & S. Schecter, 2007 USENIX Security) and "Lessons Learned from Five Years of Building More Secure Software" (M. Howard, 2007 MSDN Magazine) where Howard wrote of his experience at Microsoft: "No other metric that we measure is as valuable when prioritizing code review - not code complexity, not line number count, not code churn. The number-one indicator of potential vulnerability density is simply the age of the code."
18 http://www.gnu.org/software/freeipmi/manpages/man8/rmcpping.8.html
19 I amplified about this at http://fish2.com/ipmi/dell/secret.html
20 Not only were there bugs that were obvious to a casual observer, but seemingly there was no thought to security; for example almost no effort was used to filter input from untrusted sources.

[21] *Tcpdump* is a powerful packet sniffer that can listen to network traffic, and *gdb* is a debugger that may be used to start and stop programs, step through program execution, dump memory, etc.
[22] 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.
[23] Monitoring is possible, of course, but typically done sporadically and for trouble-shooting and performance reasons, not typically for security.
[24] Perhaps there is some large enterprise that can make it work, but the larger you are the harder it gets.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] 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.
[30] 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.
[31] 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.
[32] 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.
[33] 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.
[34] The BMC has the potential to read and write directly to them as well.
[35] More on this in section 2, "The BMC." 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.
[36] 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.
[37] "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.)
[38] 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.)
[39] 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.
[40] 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.
[41] 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.
[42] 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.