An ISP Perspective, jail(8) Virtual Private Servers

Isaac (.ike) Levy, <ike@lesmuug.org>

Materials prepared for AsiaBSDCon 2007 Proceedings, University of Tokyo, Japan.
These materials are Copyright 2006 Isaac Levy, under the terms of the BSD license.

The denial of complexity is the beginning of failure.

- Swiss historian, Jacob Burkhardt

..with proper design, the features come cheaply. This approach is arduous, but continues to succeed.

- UNIX co-creator, Dennis Ritchie

...As in all Utopias, the right to have plans of any significance belonged only to the planners in charge.

- Jane Jacobs, *"The Death and Life of Great American Cities"* [0]

One of the first significant elements of UNIX [1], was process time-sharing [2]. It's easy to forget these early times, as we now commonly touch relatively inexpensive multi-cpu hardware, eclipsing the power of a PDP-11; with smp and multi-threading kernels. Computers therefore manage simultaneous processes scaled to levels only the most adventurous could dare imagine back when UNIX first appeared. Active and persistent memory have of course scaled with raw CPU power. And it continues to get faster. We all know this.

We all know about machines, and have come to repeat the design intentions of time-sharing in many forms, including the FreeBSD jail(8) facility- a *virtual* machine.

The jail(8) subsystem in FreeBSD is well known to be an incredibly secure and durable system for partitioning processes, memory, network, and disk i/o. Building on the simplest of core UNIX subsystems, jail is an elegant base for creating Virtual Private Servers (`# man 8 jail`) To bastardize this rich and elegant system on FreeBSD:

chroot(2), bound to an **IP address**, minus some relevant system calls **= jail**

(Simply add a BSD userland, and a full virtual system is born, with a confined root!)

This material assumes the reader is familiar with the jail(8) utility, and generally familiar with the mechanisms of the underlying jail(2) system call. Further reading on the use and implimentation of jail(8) can be found in the paper written by jail's original author, '*Jails: Confining the omnipotent root.*', (PHK/ Watson, FreeBSD Core) [3].

This material aims to share real-world experiences running massively jailed systems, from a ISP perspective. Diverse goals and agendas can be liberated by applying modular, self-contained, and disposable technologies- (in short, traditional UNIX principles).

Audience for these materials:

- UNIX System Administrators with demanding users, *and limited hardware resources*

- Internet Service Providers who wish to provide robust shared hardware services

- Internet Service Providers with rigorous high-availability requirements, where mutually untrusted users and processes pose a threat to service reliability (uptime)

- Institutions with fast-paced development, learning, or short-lived server requirements

The iMeme Experience, my time at a small jailing ISP- (the first of it's kind?)

Around 2000 I became a customer at a small web hosting company called iMeme. The iMeme specialty, root-access virtual servers (using FreeBSD jail(8)). My need, was to run and further develop the behemoth web application server, Zope. I needed basics- root, a compiler, cron, logfile analysis and reporting tools- (a full server). My budget was under $70/mo usd, and back then a dedicated server was unrealistic at that rate- I needed virtual-hosting scaled prices.

By 2002, iMeme hit some stiff 'problems' when a partner left, I was then asked to join the company- and we gave it quite a go. During my time at the company we hit a mark of 1000 domains hosted, in around 470 jailed systems. The ISP was unique in that once you paid for your jailed system online, it was 'booted', and you had access to your new server- no Administrator action was necessary. iMeme, as a company, later died based on external business problems.

Mutually Untrusted Users, (and processes).

2007, it can be estimated there are 785 million people using the ipv4 internet [4], arguably a critical mass. Most of these users have personal computers, yet a great deal of computing today, again, happens on servers, offering services in various contexts.

As the needs of users become more sophisticated and varied, the applications become a uniquely fragmented environment. From a birds eye view, an astounding amount of computing machinery makes all these network applications run. From a micro view, it doesn't take much computing machinery to run a single Gmail account- (from the CPU clock perspective).

With that, the proliferation of network software which looks suspiciously like 'websites', (and perhaps mislabeled as such), are starting to to take various business applications off the PC, and onto the webserver, en masse'. Everything from content and asset management systems, to financial accounting and transaction systems, to the core of the internet- information exchange through blogs, online communities, and on, and on. Through a sort of promiscuity of form [5], http applications are evolving to manifest timeless forms of 'traditional' software.

Users of any given ISP always include developers, hackers [6] , us. The mass of internet users who do not hack, have the same sophisticated and diverse demands. For example, thank MySpace for escalating user expectations in mass-market accessibility in http server applications. With that, iMeme aimed to provide an inexpensive base platform for new internet applications like this to grow.

The real world of iMeme users: A hacker: "I want to compile LISP", An undergraduate sociology student: "I want to install 'Foo' blog software, it's PHP and the instructions say I need to run Cron", A web designer: "I want to run an http server on port 8080". A business owner: "I want to run Foo web application for my business." A community leader: "I want to run Mailman List Manager", A 13 year old hacker: "I want to run both an IRC and jabber

server for my friends". Most iMeme users simply, just wanted to hack Python/Zope.

Fairly simple requirements, yet so hard for commodity web hosting to accommodate!

Each of these users demands, and deserves, root.

The real world of iMeme users was extremely diverse. From a business perspective, the 'markets' served were all considered niche- hosting companies thought we were crazy. However, we felt the internet is merely niches stitched together to make a whole, and jail enabled a unique opportunity to build our ISP in the model of a metropolitan city [7].

Timeless Methodology in Computing (UNIX, the undead in computing)

Ancient UNIX computing models revolved around a model which the PC era did away with: server applications, feeding thin clients (server + many UNIX terminals). PC's evolved, and network computing became largely a peer-to-peer affair. The internet, has now brought a swing in the pendulum back to thin clients, as the Web Browser, as software, takes on the same role a terminal did years ago- and UNIX is right there, ready and waiting to handle the applications- with an astounding wealth of time-tested (and some ancient) tools well suited for managing multi-user multi-process servers.

With that, simple, modular, disposable utilities are vital to meeting the diverse needs of the iMeme user, in providing a full Virtual Private Server environment.

When jail(8) was first introduced to FreeBSD, it was (and still is) a simple utility, written in the spirit of old UNIX. As a simple utility, jail(8) provided iMeme the opportunity to build on the work of others and avoid reinvention and incompatibilities, (classic UNIX methodology).

jail(8) therefore proved itself well suited to to taking on the complexities of our user needs, which were essentially limitless. Other Virtualized system designs come close, but insomuch as most Virtual OS systems take on the monolithic responsibility of providing all system interfaces, (virtualized memory, networking, filesystem), they all critically failed to meet the iMeme needs in one area or another- as their respective histories were to meet a particular computing problem, or use case.

The history of computing is littered with the corpses of Virtual OS systems, all of which end up withering under the sheer weight of the computational responsibilities they take on. However, like UNIX time sharing, simple and modular components of computational virtualization seem to be the only elements which persist. Subsystems like UNIX users and ACL's, actually the entire concept of UNIX privilege separation, follows in the footsteps of the simple mechanism of time-sharing. Enter, jail(8), 1998.

As a small and complete utility, jail(8) is much like the invention of of the Otis Elevator and it's affect on the design of skyscrapers,

"In the era of the staircase all floors above the second were considered unfit for commercial purposes, and all those above the fifth, uninhabitable." [8]

The jail(8) utility, enabled the same sort of liberation of space, and with the same overtones of 'safety'- if one compares security features to elevator safety concerns, (falling).

(Running the risk of sounding silly, I am directly comparing an internet hosting ISP to a skyscraper, and skyscrapers are different from other types of buildings.)

The iMeme Experience (System Specifics)

The iMeme systems were quite simple for UNIX administrators to understand.

We ran high-density 2u (and then 1u) servers, which we aimed to have approximately 50 jails running on at any given time. In 2001, a base account was provisioned 4gb of disk space, and 100mb of what we called 'process space', the amalgamation of memory and cpu usage. Bandwidth was rarely an issue worth metering back then, so very basic QOS oriented throttling was performed to ensure every user had a fair slice of available network traffic.

For disk space, we ran scripts from the host server which simply used du, and shoved the output into MySQL databases- where we then automated the process of implementing policies of charging for extra disk usage. We choose to give 1 month of 'grace time', insomuch as sometimes logfiles would explode, or users would accidentally consume undue disk space- and we felt this was a simple buffer our customers appreciated.

Hard limits for disk space were always a consideration. Disk slices were far too rigid to meet user demands, (creating extreme overhead in managing upgrading disk space), though we did experiment with them. A persistent risk was that a user, by choice, accident, or compromise, could consume all the available disk space for a jailing system. With that, again, simple unix strategies came back into place to contain the problem. The strategy we ended up liking best was to absolutely a partition for jails, (the majority of available disk), and then perhaps break it into a few chunks to isolate various jailed disk space from each other. After time, 80gb slices worked nicely, and fitting 4x 300gb drives into 1u, this afforded a sort of 'neighborhood' partitioning. Extreme cases of disk consumption were further restricted on a per-case basis, using file-backed memory disks (disk images); **but,** especially in recent FreeBSD releases, this incurs an additional i/o penalty, which users do not appreciate- (and it soaks RAM on the host system as well). Disk images are not necessarily a practical solution for every jailed system, however flexible they are in providing hard limits to disk space.

Memory and CPU usage was polled on a regular basis for each jail. Shell scripts were originally setup to run as cron jobs *inside each jail*, which took cumulative memory consumption and cpu usage by parsing ps(1) output inside a given jail. While iMeme originally ran thes scripts inside of each jailed system, outputting totals to text files in /jail/dir/var/log/, however this always carried the risk that a user could (trivially) bypass this system to avoid increased billing or otherwise. In their jail, remember, the user has root. That stated, eventually iMeme moved this system out to the host system with new jailing features in FreeBSD 5.x- insomuch as one can list/kill processes based on the jail id, information availble to ps, and processes listed in the /proc filesystem.

FreeBSD 4.x jailing relied heavily on a jailed hostname for host-level process identification (and subsequent management)- which created problems. If a user changed their hostname, accidentally or maliciously, havoc would follow for management systems in the host system. FreeBSD 5.x solved this problem by pinning a 'jail id' to each process on the system, and providing a sysctl to lock down the ability to change hostnames within a jail.

Jailed process restrictions were then handled neatly using renice(8). Processes which hogged undue CPU were simply renice'd by the host server, releasing the process renice level after 5 minutes to see if the process was again behaving. If not, it was reniced again. This crude strategy was wildly successful in maintaining fair-share cpu and memory usage for processes. Problem processes, (things with memory leaks, for example), were then in the hands of the jailed user to deal with- without negatively impacting the other jailed users.

Fork bombs were still a threat, but from FreeBSD 5.x onward, each jail could be set

to start with an escalated securelevel, and maxprocs could be locked for a jail, chflags(2) disabled in jails via host sysctl settings, and viola- fork bombs as a threat are mitigated, with relatively minimal management and resource consumption.

Network resource management is far outside the scope of this material, however, it is worth mentioning one thing: at iMeme, each jailing hardware server was conceptually treated like a network border or gateway, with routing and filtering tasks carried out inside the machine. This paradigm shift in management greatly simplified the physical network requirements, (making routers, firewalls, nonexistent). With that, we ran NAT for our external IP blocks, and mapped addresses to our jails- which all ran using a private netblock, (192.168.x.x). This NAT strategy had pros and cons and is hardly worth discussion- except to state it all was run from the host servers, with negligible impact on jailed systems. Also, back then, ipfw(8) and dummynet(4) were used for very minimal network management- dummynet(4) configured to provide eqal-share bandwidth (ad-hock QOS), and IPFW was crudely used to put out fires. Today, in my Diversaform jail cluster, pf(4) nicely replaces these tools- and is becoming the de-facto packet filter- and in 5 more years, there may be something else, but it will still be running from the jailing host hardware.

Large Scale Management Techniques
(System Specifics)

At iMeme, we maintained Master Record Server (obviously a redundant system). This system primarily kept the MySQL database which recorded everything from resource usage, to billing and contact information. This strategy worked well, provided any modifications/additions to this system were thoroughly tested. This was easy, insomuch as we could replicate this system in one of our jails at any time, and then dispose of the jail. There was no reason in particular for the MySQL database, it was just used in the beginning and stuck with us reliably.

The website, where users bought jailed systems, and managed their account and billing, was all written in Zope, and had PHP elements added over time. This could have been any web technology.

As each iMeme jailed system had some custom tweaks, we maintained a pre-compiled FreeBSD useraland, preconfigured with any small tweaks to our enviornment (like the cpu/memory polling cron job mentioned before). These jailed systems were built, and put into cvs(1) repositories for long-term management, however tar(1) became the deployment tool of choice. Scripts to add new systems would effectively untar the current jailing userland, and then run scripts to add an initial user, add the root password, and start the jail.

Upgrading jails was a trivial technical process. System upgrades were handled similarly, un-tarring updated userland sources to jailed userland directories. Following the hier(7) man page, users additional applications ended up in /usr/local, and only in extreme edge cases did a customer application have problems with minor dot upgrades, (4.5 to 4.6, for example).

In FreeBSD 5.x, it became clear that running installworld, and tossing it an additional flag for the jailed directories, was even simpler than the tarballs, with the additional benefit of dispensing with keeping userland (binaries!) in CVS.

When monitoring the systems, based on the rapid scaling possibilities with the ease of adding jails, keep monitoring simple- and quiet. When problems occur on jailed systems, it's *always* possible that all jails on a particular host are affected, so if they all trip alarms, administrators can get lost in white noise. An experiment, was logger(1)/syslog(3). iMeme tried pushing all jailed logs out to master syslogd(8) server, with nearly

worthless results. The valuable information was covered by the white noise of everything users were doing and running in their systems, and it also provided outright surprising breaches of privacy- so iMeme abandoned this idea immeadiately. While there are ways to sanely utilize syslogd(8) schemes, they are far outside of the scope of this material.

Jailing Redundancy (failure is life)

Jails present a uniquely simplistic mechanism for backup and fail-over. At iMeme, each jailing host kept jails in /usr/local/jails. As time and internal methodology evolved, (disk slice strategies, etc...) /usr/local/jails/hostname.jailing.host became collected mount ponts and soft links, but the userland interface was always the same to find a given jail: /usr/local/jails/hostname.jailing.host/JAIL_DIR

Then, each jaiing host both exported, and mounted, all other jail directories as an NFS mount. This carried extreme management benefits, worth the hassle and cursing associated with heavy NFS use. Operations could be carried out on each jailed userland *from any jailing host in the cluster*! With that stated, backups and restore became simple operations. Backing up became an operation of tarballing each jail to a backup server, (independently redundant), and restores consisted of untarring the jailed userland in the NFS mount of a jailed host. If a jailing host server died, all of it's jailed systems could then be rapidly re-distributed and re-started across the whole cluster. This process required Administrator intervention.

Post-iMeme, Diversaform jailed systems are run slightly differently- without NFS. Each jailing host has an identical hardware machine, which jailed systems are regularly synchronized to. If a jailed application requires time-based backups, it is synchronized to another jailing server (itself having a hardware twin). Diversaform systems have also been experimenting with a combination of carp(4) and ggated(8) (GEOM Gate), providing network interface virtualization and network block-level disk mirroring, but due to discovered inconsistencies of the FreeBSD carp(4) mechanism, and the relatively low adoption (and documentation) of ggated(8), this setup is still considered experimental. However, as these tools mature, they promise to help bring real-time failover of jailed systems- *without* Administrator intervention.

One last strategy for jailing failover has been called 'The Golden Jailing Formula': NFS mass storage backing for jails' userland, running on thin servers in a cluster. This is an excellent strategy, excepting it's restrictiveness for scaling. Many jailing administrators have attempted this formula, yet it doesn't scale as modularly as the iMeme strategy- which uses many jailing hosting servers, (or one jailing host). Total storage, i/o throughput, and then redundancy of this system make scaling jails difficult- and the reality of jailing, is that in many contexts, jailed systems grow far beyond initial expectations. So while this is a technologically viable plan, social, political, economic, and human factors limit it's success in most manifestations of massively jailed environments.

User Segregation (a bad idea)

Back to the various mutually untrusted users, a component of any massively jailed systems environment is to segregate users according to their threat level to the whole. This quickly takes jailing into philosophical approaches to social, political, economic, and human factors.

The wily hacker is your friend. iMeme founders' roots literally grew up at the annual Defcon security conference, in the USA. With that, many iMeme customers were politely put, a bit insane- and very demanding. Should these users be identified and placed on their own hardware, so some hackhing hyjinks don't get out of control and affect the

'small business' or 'nice' customers? This is a common question iMeme wrestled with.

However, at this real-world massively jailed ISP, the very opposite scenario manifest. The 'small business' user often followed less than adequate security practices, as well as running less stable software. With that, it was more often the 'wiley hacker' complaining about their small-business or blogging neighbor.

Regardless, it became clear that there was no viable metric for how or when to segregate users to given hardware servers , and in the end it became irrelevant in mitigating the risks inherent in any shared system. So what did iMeme do? This problem is a constant, use this environment to advantage for all.

Blindly distributing types of users across all hardware, had the distinct advantage of leveraging everyone's shared needs- keeping all systems online. The use of a customer/ community mailing list created an enviornment where business owners and wiley hackers alike, could share experiences and discuss problems- all with the common aim of solving the problems. Additionally, this community took a great deal of impossible administrative overhead out of iMeme Administrator hands. It helped set the expectation that we just ran the servers, but had no expertise in using FOO PHP blog software, or BAR irc server, etc...

That stated, hackers who monitor uptime were a guard for the 'business owner' who did not, and the various social and cultural diversity of the user base ensured *somebody* was online 24/7. While this made for excellent catch-all systems monitoring, it also made it difficult to schedule upgrades- a trivial point in the context of the benefits.

Developers vs. Production Users is likewise a poor segregation line, insomuch as 'developers' are often hammering systems that 'production users' may rarely touch- and can help spot system problems before they become critical (arbitrary inode corruption, for an anecdotal example).

In the end, user diversity decreased overall failure risks in iMeme systems.

Conclusion

Through a combination of building on thirty years of UNIX, attention to social concerns, and respect for undeniable complexity, jail(8) was leveraged to great success at the ISP iMeme.

The most valuable elements in successfully running massively jailed systems were not cutting-edge technologies, but the application of ancient practices in computing, urban design, and to a great extent social and political sciences.

Now that iMeme is gone, who's next? What ISP, in private, commercial, or other contexts will step foreword to provide virtual systems?

Doesn't everyone deserve root?

Awknowledgements

Jon Ringuette (wintermute),
founding partner of iMeme, to it's end.
<bobskr@gmail.com>

Ethan and Jake, iMeme founding partners

Dave Lawson, (reality), iMeme core friend

Elise, the P1rate, whit537, beren1hand, and all the iMeme community who LIVED on irc

Footnotes

Note: parens in the text, ( ) refer to a corresponding UNIX man page, notatin of brackets [ ] refer to the footnotes below.

[0] Jane Jacobs , '*The Death and Life of Great American Cities*' Copyright 1961 Jane Jacobs, Renewed 1989.

[1] D. M. Ritchie and K. Thompson , '*The UNIX Time-Sharing System**', First presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15-17, 1973.

[2] L. P. Deutsch and B. W. Lampson, `SDS 930 time-sharing system preliminary reference manual,' Doc. 30.10.10, Project GENIE, Univ. Cal. at Berkeley, April 1965.

[3] Poul-Henning Kamp <phk@FreeBSD.org> , Robert N. M. Watson <rwatson@FreeBSD.org>, '*Jails: Confining the omnipotent root.*' The FreeBSD Project

[4] World population is 6.45 billion today according to World Gazetteer <http://www.gazetteer.de/home.htm> ;it was 6.3 billion in 2002 according to the UN, <http://www.un.org/esa/population/publications/wpp2002/WPP2002-HIGHLIGHTSrev1.PDF>. Internet World Stats says 785M (March 2004) <http://www.internetworldstats.com/stats.htm>. NUA said 606M (Sept. 2002) <http://www.nua.ie/surveys/how_many_online/>. A claim that "roughly" or "about" 10% of the world's people have Internet access seems safe.

[5] Jonathan Crary, *J. G. Ballard and the Promiscuity of Forms*, Zone 1/2 (1986), pp. 159–65. 38- also reprinted in various MIT press publications.

[6] Hacker is mostly used in the Eric S. Raymond (ESR) sense of the word- "The term 'hacker' also tends to connote membership in the global community defined by the net" <http://www.catb.org/~esr/jargon/html/H/hacker.html>

[7] Christopher Alexander, '*A City is Not a Tree*', (1968), Originally published in: Architectural Forum, Vol 122, No 1, April 1965, pp 58-62 (Part I), Vol 122, No 2, May 1965, pp 58-62 (Part II). Available online in C. Alexander's Archives, <www.patternlanguage.com>

[8] Rem Koolhaas, '*Delirious New York: A Retroactive Manifesto for Manhattan*', (1997), Monacelli Press

Without the amazing support and existence of the following institutions, none of this material would have been possible :

NYCBUG
NEW YORK CITY *BSD USER GROUP

diversaf■rm inc.