;login:

THE MAGAZINE OF USENIX & SAGE
April 2003 • volume 28 • number 2

2003 USENIX
SPONSORED BY USENIX, THE ADVANCED COMPUTING SYSTEMS ASSOCIATION
Annual Technical Conference
JUNE 9–14, MARRIOTT RIVERCENTER, SAN ANTONIO, TEXAS
Don't miss out!

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

USENIX Upcoming Events

The First International Conference on Mobile Systems, Applications, and Services (MobiSys '03)

Jointly sponsored by ACM SIGMOBILE and USENIX in cooperation with ACM SIGOPS

MAY 5–8, 2003
San Francisco, California, USA

http://www.usenix.org/events/mobisys03/

The 9th Workshop on Hot Topics in Operating Systems (HotOS IX)

Sponsored by USENIX in cooperation with IEEE TCOS

MAY 18–21, 2003
Lihue, Kauai, Hawaii, USA

http://www.usenix.org/events/hotos03/

2003 USENIX Annual Technical Conference

JUNE 9–14, 2003
San Antonio, Texas, USA

http://www.usenix.org/events/usenix03/

ACM/IFIP/USENIX International Middleware Conference

Sponsored by ACM, IFIP, and USENIX

JUNE 16-20, 2003
Rio de Janeiro, Brazil

http://middleware2003.inf.puc-rio.br/

2003 Linux Kernel Developers Summit

JULY 11-22, 2003
Ottawa, Canada

10th Annual Tcl/Tk Conference

Sponsored by Noumena Corporation, in cooperation with ActiveState, USENIX, and IEEE SouthEast Michigan Computer Chapter

JULY 29-AUGUST 2
Ann Arbor, Michigan, USA

http://www.tcl.tk/community/tcl2003/

12th USENIX Security Symposium

AUGUST 4–8, 2003
Washington, DC, USA

http://www.usenix.org/events/sec03/

Camera-ready final papers due: May 13, 2003

BSDCon '03

SEPTEMBER 8–12, 2003
San Mateo, California, USA

http://www.usenix.org/events/bsdcon03/

Notification of Acceptance: May 12, 2003

Camera-ready final papers due: July 8, 2003

5th IEEE Workshop on Mobile Computing Systems & Applications

Sponsored by the IEEE Computer Society, in cooperation with ACM SIGMOBILE and USENIX

OCTOBER 9-10, 2003, Santa Cruz, CA, USA

Web site: http://wmcsa2003.stanford.edu

17th Systems Administration Conference (LISA '03)

Sponsored by USENIX and SAGE

OCTOBER 26-31, 2003
San Diego, CA, USA

http://www.usenix.org/events/lisa03

Paper submissions due: April 21, 2003

Third Virtual Machine Research and Technology Symposium (VM '04)

MAY 6-7, 2004, San Jose, CA, USA

Web site: http://www.usenix.org/events/vm04

Paper submissions due: October 13, 2003

For a complete list of all USENIX & USENIX co-sponsored events, see
http://www.usenix.org/events

contents

;login: **Vol. 28 #2, April 2003**

;login: is the official magazine of the USENIX Association and SAGE.

;login: (ISSN 1044-6397) is published bi-monthly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$50 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $110 per year.

Periodicals postage paid at Berkeley, CA, and additional offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

motd

by Rob Kolstad

Rob Kolstad is currently Executive Director of SAGE, the System Administrators Guild. Rob has edited *;login:* for over ten years.

kolstad@sage.org

I have some comments on some of the recent press reports about "Cyber Terrorism" and computer security in general that contain fabulous quotes that the media has widely disseminated.

James A. Lewis, a Center for Strategic & International Studies analyst wrote a monologue entitled, "Assessing the Risks of Cyber Terrorism, Cyber War and Other Cyber Threats"[1]. Simply, he said:

> The assumption of vulnerability is wrong.

and in the paper's stated context, he's right. Its second paragraph defines: "Cyber-terrorism is 'the use of computer network tools to shutdown critical national infrastuctures (such as energy transportation, government operations) or to coerce or intimidate a government or civilian population." OK. It's fair to define terms and then analyze the results of that definition.

The Gartner Group published a Q/A column[2] entitled *Cyberattacks and Cyberterrorism: What Private Business Must Know*. Here's a quote that I think would be warmly received by those trying to decrease IT security budgets:

> Criminal cyberattacks are real and occur daily, while cyberterrorism is still a theory. Despite the hype, there is no known case of cyberterrorism. Government efforts focus on helping enterprises help themselves, and each other.

and, later:

> [The] Federal Bureau of Investigation defines terrorism as "unlawful or threatened use of force or violence . . . against persons or property to intimidate or force a government [or] civilian population [to further] political or social objectives." GartnerG2 defines cyberterrorism as "terrorism attacks using a digital channel."

Defining words is important. I think that defining words carefully and then using them publicly in quotes like "Cyberterrorism is still a theory" advances ideas that are counterproductive and can easily lead to decision-making with wide-ranging negative repercussions.

The recent Slammer Worm, 376 bytes of code that flashed through the internet recently, demonstrates one dimension of the state of the "cracking" art. The CAIDA folks have published an analysis called "The Spread of the Sapphire/Slammer Worm," principally authored by David Moore. It contains these facts:

- The number of infected systems doubled every 8.5 seconds during its first minute. That's 133x growth in a single minute.
- **At its peak (3 minutes in), it scanned 55,000,000 systems/second** (that's a rate of 3.3 billion per hour). After that, no more network bandwidth was available. This is 100x the speed of the July 19, 2001 Code Red Worm, which hit 359,000 hosts.
- Of all systems possible to infect, 90% were infected within 10 minutes of the worm's launch.
- The worm began to infect hosts around 05:30 UTC on Saturday, January 25, 2003 and exploited a buffer overflow discovered in July, 2002. A patch was released shortly thereafter, months before the worm was launched.
- 75,000 total hosts were infected.
- **A "better" vulnerability would have enabled infection of the entire internet in 15 minutes**, a "flash worm" or a "Warhol Worm."
- Neither Code Red nor Slammer had a malicious payload.

Now, this sort of result doesn't agree with Joshua Green comments for *Washington Monthly* in November of 2002[3]:

> There's just one problem: There is no such thing as cyberterrorism – no instance of anyone ever having been killed by a terrorist (or anyone else) using a computer. . . . What's more, outside of a Tom Clancy novel, computer security specialists believe it is virtually impossible to use the Internet to inflict death on a large scale, . . .

Dorothy Denning, Georgetown University Professor and cybersecurity experts says, "Not only does [cyberterrorism] not rank alongside chemical, biological, or nuclear weapons, but it is not anywhere near as serious as other potential physical threats like car bombs or suicide bombers."

The article goes on to cite US$15 billion lost last year to various cyber attacks.

I believe that all these authors are writing words that are true: No one (hardly anyone?) has died from a computer attack, at least directly. The Slammer worm did infect some 911 computers; it's not apparent that any real problems emerged from that.

But I think that all these analyses are missing the point. By demonstration, the computer and networking infrastructure is vulnerable. Slammer was a small worm with a relatively low success rate.

The networking infrastructure drives huge parts of the USA and world economies. It's easy to see that the fallout from the 9/11 attacks has contributed to the bankruptcy of airlines and, I believe, contributed to the lengthening of the USA economic recession. The attacks and their media coverage have moved USA society, at least, to a state of paranoia and fear not seen since the nuclear bomb scares of the late 1950s and early 1960s.

Please join me in a mental exercise about cyberattacks. Imagine that:

- a cyberattack causes the banking system to fail in its interbank money transfers.
- a cyberattack infects enough routers on the internet that traffic is slowed by 100x for weeks as the routers ping-pong infect one another.
- a cyberattack infects so many business computers that deliveries of commercial good of all kinds (including food) are halted for a week and then restarted manually with concomitant inefficiencies and lower throughput.
- a cyberattack takes out the airline reservation system, causing airlines to suspend their operations.
- a cyberattack takes out the computer control of the communications infrastructure, thus reducing telephone and other bandwidth by 100x and disabling long-distance phone calls, internet connectivity, and 80% of television programming relays

While none of these results would result in widespread death, any one of them would foment panic, chaos, and a deep new fear about the stability of at least the USA economic foundations and maybe its society in general. This level of disruption has the potential to destroy huge portions of the commercial infrastructure and unemploy millions of workers. It is easy to believe that the results would dwarf the depression of 1929.

Let's take a rational and calm course in "selling" computer security. Let's not use "Fear, Uncertainty, and Doubt" to motivate decision-makers. But please: let's not stick our heads in the sand, either. The Slammer Worm is only the latest realized threat. It is difficult to believe that no more threats exist. Keep your systems patched; heed vendors' warnings about upgrades; use common sense security precautions when designing and procuring software. Let's not make attackers' goals easier to achieve.

1. *http://www.csis.org/tech/0211_lewis.pdf*

2. *http://www.gartnerg2.com/qa/qa-0902-0091.asp*

3. *http://www.washingtonmonthly.com/features/2001/0211.green.html*

comments on the national strategy to secure cyberspace

by Dan Geer

Dan Geer is a USENIX Past President and is Chief Technology Officer at @Stake, Inc.

geer@world.std.com

The Critical Infrastructure Protection Board (CIPB) released the Draft National Strategy to Secure Cyberspace for comment last November. The strategic shift it proposed was to focus on vulnerabilities rather than opponents. While to a *;login:* audience it may well seem trite to note that on the Internet every sociopath might be your next-door neighbor, in endorsing this idea the National Strategy broke new ground for audiences other than ours. As the critical infrastructure of this and any other free nation will be dominatedly in private hands, policy at the national level will flow through people like us whether we are design-side or operations-side, USENIX or SAGE. To give one view on what this will take, what it will mean, what it will demand, below you will find my own formal response to the National Strategy, as is and as delivered. Formal responses such as this have not yet been made publicly available and so I cannot directly point to the body of responses in full. I can say that all of us here have a role to play one way or another in a world where what we do is already critically necessary. I urge all of you to be involved at whatever limit of skill and wisdom you possess.

Note that the situation is moving faster than *;login:*'s publication schedule can handle–as of this writing (early March) the final draft of the Strategy came out and the CIPB was disbanded shortly thereafter, leaving lobbyists torn between how-wonderful-no-regulator and how-terrible-no-grant-money emotions.

Provable security is never affordable while affordable security is never provable

To:	Richard Clarke, Howard Schmidt, et al.
From:	Dan Geer, CTO @stake, and other affiliations
Date:	18 November 2002
Re:	Primum non nocere – National Strategy to Secure Cyberspace

Because (1) the cost of duplication of electronic information is zero and (2) the Internet is a commons where distance in space and time is irrelevant, it is therefore absolutely necessary to replace a military style doctrine of security (focused assessment of hostile parties and their capabilities) with a market style doctrine of security (focused risk management of one's own vulnerabilities). In this we concur with the Draft National Strategy.

Provable security is never affordable while affordable security is never provable; tradeoffs are therefore natural and inevitable. Such tradeoffs are preferably based on market forces except where market failure is intolerable. Such market failure is likely whenever risk is diffuse and deferrable. Risk will seem diffuse and deferrable in an absence of adequate visibility to the sharable risk information we collectively do have or where whether to act or not is based only on qualitative argument. The Draft speaks to an

The single most fundamental enemy of security in cyberspace is complexity.

"information deficit" and indeed there is one. The top priority going forward is to share data that is already in hand and to share it in a way that provides coherent, quantitative decision support. Unless and until there are quantitative measures, a market failure is unavoidable. The need for legislative relief in this area, e.g., the Bennett-Kyl "Critical Infrastructure Information Security Act," is real.

The information technology woven through our critical infrastructure is what in a military setting would be called a force multiplier–it adds significant power to its user out of all proportion to its cost, ergo, it is about productivity. When a force multiplier is developed by and for an elite, it will be used in whatever way the goals and the constraints, the culture if you will, of that elite would imply. When that force multiplier is made available to everyone regardless of their goals and their constraints, it should come as no surprise that goals and constraints may well have to be injected – through the rule of law, one should hope. That there is a need for a National Strategy to Secure Cyberspace at all confirms this observation.

The single most fundamental enemy of security in cyberspace is complexity. Any solution in the name of security that increases complexity is asking for trouble; any solution that costs more in productivity than it produces will consume wealth. As such, security solutions must be simple or they will be unsustainable – security's accumulating costs will grow in visibility as security's deliverables will shrink in visibility when recent events inevitably recede in group memory.

Networked communications are increasing in type, kind, and capacity to the point that the perimeter of every entity, public or private, is dissolving. When perimeters cannot be defined they cannot be defended, and when they cannot be defended there is no practical difference between the "inside" and the "outside" of the organization or society. Plainly put, perimeter defense strategies are trending sharply toward the diseconomic and will eventually fail. The effective blurring of inside and outside, exaggerated by the cost trend made above, makes security in the form of "access control" long-term unsustainable. When access control is unsustainable, the only alternative is accountability.

Accountability for actions taken is the hallmark of a free society. Accountability requires records. When the nature of offenses to security can be enumerated a priori, the records on which accountability will be based can be minimal and their handling can be procedural. When the nature of the offenses to security cannot be enumerated a priori, neither can the records that would otherwise be required to enforce accountability a posteriori.

The form of accountability most consistent with existing private sector structure and habit is that of liability. Just as the Draft National Strategy calls on all levels of society to do their part, the need for accountability must extend to all levels of society. That includes accountability in the form of liability, but if and only if that accountability is built on the calculus of risk rather than the calculus of influence. The two principal areas where the lack of teeth in the Draft must be addressed are both of this sort: product and service vendors who sell insecure products must shoulder the liability therefrom while companies and persons whose property is used for attacks on others must shoulder the liability therefrom. The easily confirmed persistence of known, fixable security flaws in the field already proves that the present regime of information-push without liability-based accountability simply will not work. Conclusions include but are not limited to:

- Software vendors can make security claims their customers are in no position to confirm. Hence, software vendors must not make security claims without either assuming the liability for those claims or by seeking, for the claims, the confirmation of a third party audit.
- Computer network operators are already well motivated to police and repulse inbound attacks. Hence, computer network operators must become liable for attacks outbound from their networks.
- When infrastructures, fail they do so in cascade. Management of large computing infrastructures is made easier by homogeneity within that infrastructure but the only defense against common mode cascade failure is heterogeneity. Hence, infrastructures that choose homogeneity must separately correct for the added risk burden they thereby impose on those who depend upon them.
- Security is a means; it is not an end. With rapid technical change, means cannot be durable. Only ends can be durable. Security-centric "procedural correctness" without a concrete description of goal state(s) makes any risk worse by soaking up the resources that might actually have gone to genuine repair and by comforting the ill-informed. Hence, whatever regulation is created or augmented in support of the National Strategy must be about ends and not means.
- The critical infrastructure of the United States of America obviously includes private sector firms that are not US-based. Hence, the National Strategy will not complete so long as it is bound to US-based firms only and the National Strategy must be so modified.
- Because the price of bandwidth falls faster than the price of storage which falls faster than the price of computation, long-term economic pressures favor capturing observable events to multiple locations for later analysis when and if needed. Because a linear growth in network end-points creates a geometric growth in network complexity, any entity charged with security must never fall behind in data collection even if it assumes analysis of that data may never occur or occur much delayed. As observability in the electronic sphere vastly exceeds observability in the physical sphere, hence the phrase "expectation of privacy" must be based on cultural norms as it cannot be any longer based on what is within the realm of the possible.

In sum, details, quantitative details, matter. We, all of us, must share data on occurrence of risk and countermeasures to risk, we must share it in a setting of sticks and carrots that harness self-interest rather than work against it, and national policy can only stress quantifiable ends that must be met or liability be assessed, assuming that the National Strategy to Secure Cyberspace wishes to itself reach a goal rather than merely look good.

Security is a means; it is not an end.

IPv6 configuration on Solaris 9 and freeBSD-4.x

by Timo Sivonen

Timo Sivonen is a principal consultant working for Greenwich Technology Partners. His professional activities circulate around operating systems, internetworking, and information security.

tljs@greenwichtech.com

The basic setup to enable IPv6 on the Solaris 9 and FreeBSD-4.x operating systems is discussed. The text will begin with a quick overview of the IPv6 packet format and various addressing types, before continuing to stateless address autoconfiguration for a host and a router. The text will also deal with the current and transition issues of the Domain Name Service for IPv6. Finally, there is discussion about IPsec and how to use IPsec for inbound authentication and to secure communication between two hosts.

Introduction

"Is there a customer requirement?" and "Have you been asked about migration?" were the first comments I received when I said that I had started to work on IPv6.[1] Undeterred by reality, my incentive to play with IPv6 was to experiment with the protocol to see how it works. Although the buzzwords "next generation Internet" and "migration" have been flying around for the past eight years, no one really knows whether this will ever happen. Instead, I decided to take a fresh look at IPv6 and view it as a completely new network protocol that just happens to improve IP further.

IPv6's availability has improved in the past few years. It is no longer a network protocol for only researchers to play with, but the emergence of IPv6 stacks, commercial and open source, means that IPv6 is getting serious. If IPv6 is in the operating system already, enabling it will cost nothing. IPv6 is a turnkey protocol as SPX/IPX was, and you don't have to request an IP address.

In the following text we will discuss how to set up IPv6 on Solaris 9 and FreeBSD-4.x. We will briefly cover the various addressing types of IPv6 and what they mean to an average user who wants to get his/her feet wet. A quick overview of the two recommendations for IPv6 DNS will be presented. Finally, we will discuss the use of IPSec, try a very basic setup, and test interoperability between Solaris and FreeBSD.

Getting Started — Header Format

No discussion of a networking protocol would be complete without displaying the actual packet header (Figure 1), as in RFC-2460.[1] The IPv6 header is 40 octets long, as opposed to the 20 octets of the IPv4 header. The IPv6 header is much simpler, which will speed up its processing on routers and end-nodes.

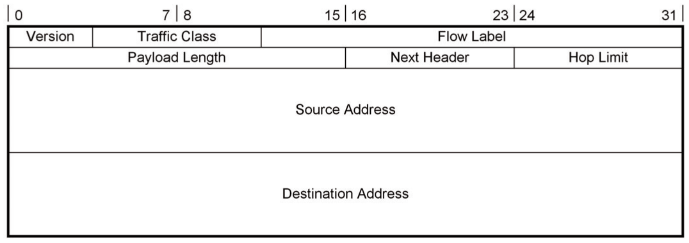


Figure 1. IPv6 Header

The fields of the IPv6 header are as follows:

Field	Bits	Description
Version	4	Internet Protocol header version. Has value of "6".
Traffic Class	8	Used by routers to recognize different traffic classes or priorities.
Flow Label	20	Requests special handling of the IPv6 packet, e.g., real-time quality of service.
Payload Length	16	Length of the payload, i.e., packet excluding IPv6, in octets.
Next Header	8	Type of header following IPv6, as in RFC-1700.[2]
Hop Limit	8	Time-to-live value.
Source Address	128	Sender's address.
Destination Address	128	Recipient's address.

The IPv6 header does not have a header checksum field. Although this seemed important in the days of IPv4, error checking in the link layer protocols has made this field obsolete. The removal of header checksum leaves the possibility that the IPv6 header could get corrupted between the network interface and the network protocol. The probability for this to happen is minimal and computers are far more reliable nowadays than they were in 1981, when IPv4 was designed.

IPv6 also has built-in provisions for Quality of Service (i.e., Flow Label, Traffic Class), although these will not be discussed in this text. QoS is a major improvement over IPv4, and IPv6 should be able to prove its worth as a voice over IP and video over IP delivery vehicle. The major advantage of IPv6 QoS is its architectural independence of the underlying link layer. Not surprisingly, online gamers have also shown some interest in IPv6.

Getting Started — Addressing

The first thing you need is an address, and IPv6 has 2^{128} of those, compared to 2^{32} of IPv4. In this section we will deal with the basics of IPv6 addresses and address formats. The logic of IP addressing has changed quite a bit from the days of Classes A, B, and C. As with IPv4, each address type is defined by its prefix and RFC-2373[3] lists the following prefixes:

Allocation Space	Prefix (binary)	Fraction of Address
Reserved	0000 0000	1/256
Unassigned	0000 0001	1/256
Reserved for NSAP Allocation	0000 001	1/128
Reserved for IPX Allocation	0000 010	1/128
Unassigned	0000 011	1/128
Unassigned	0000 1	1/32
Unassigned	0001	1/16
Aggregatable Global Unicast Addresses	001	1/8
Unassigned	010	1/8
Unassigned	011	1/8
Unassigned	100	1/8
Unassigned	101	1/8
Unassigned	110	1/8
Unassigned	1110	1/16
Unassigned	1111 0	1/32
Unassigned	1111 10	1/64
Unassigned	1111 110	1/128
Unassigned	1111 1110 0	1/512
Link-local Unicast Addresses	1111 1110 10	1/1024
Site-local Unicast Addresses	1111 1110 11	1/1024

Table 1. IPv6 Address Assignments and Prefix

Since working with binary prefixes is usually confusing and sometimes error-prone, you may want to consult the IPv6 Address Oracle[4] at the Advanced Network Management Laboratory at Indiana University. While I was trying to learn the address structure, I was almost continuously visiting the site.

Note that the dotted-decimal notation of IPv4 addresses has been replaced with hexadecimal. Practically speaking, an IPv4 address is 32 bits, i.e., 4 octets long, while an IPv6 address is 128 bits, i.e., 16 octets long. Hexadecimal representation is more economical for large addresses, and one can use "::" as an abbreviation for a string of zeros. However, this can be used only once in an address. In other words, one can write an IPv6 address as

```
fe80::280:c7ff:0:434
```

or equally

```
fe80:0:0:0:280:c7ff:0:434d
```

but `fe80::280:c7ff::434d` would be ambiguous and thus illegal.

It is possible to use the IPv4 dotted-decimal notation under certain circumstances. If there is an embedded IPv4 address inside of the IPv6 address, the last four octets are represented using the dotted-decimal notation:

```
::ffff:192.168.1.1
```

Currently IPv6 defines the following address types:

- Unicast. A unicast address identifies a single interface. There are several unicast address types such as unspecified address, loopback address, global aggregateable unicast address and local-use addresses. There are other types which are not in the scope of this discussion.
- Anycast is a special case of unicast. An anycast address identifies a set of interfaces, typically belonging to different nodes. A packet sent to an anycast address will be delivered to the nearest interface. Anycast is not in the scope of this discussion.
- Multicast. There is no broadcast in IPv6 since multicast can be used for the same purpose. A multicast address is identified by the prefix ff/8.

Unicast Addresses

A unicast address consists of the subnet prefix and the interface identifier (Figure 2). Although the interface identifier depends on the type of the unicast address and the underlying media (e.g., Ethernet LAN, point-to-point circuit or a tunnel endpoint), it must be 64 octets for all addresses in Table 1.

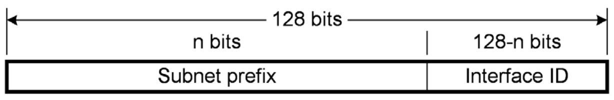


Figure 2. Unicast Address

In the following text we will discuss loopback addresses, local-use addresses and aggregateable global unicast addresses. Other address types such as IPv6 addresses with embedded IPv4 addresses, NSAP addresses, and IPX addresses are covered in RFC-2373 and Huitema.[5]

Interface Identifier

Finding the interface identifier for LAN-connected nodes is relatively straightforward, since the 48-bit MAC address is globally unique. The procedure to expand the MAC address to the 64-bit interface identifier is based on EUI-64.[6] This is accomplished by inserting two octets with hexadecimal values of 0xff and 0xfe in the middle of the address and setting the universal/local bit to 1. This is displayed in Figure 3:

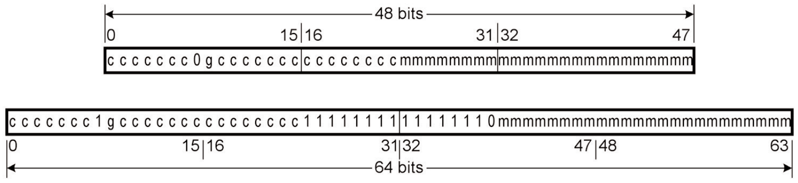

48 bits
0
15|16
31|32
47
c c c c c c c 0 g c c c c c c c c c c c c c c c c mmmmmmmmmmmmmmmmmmmmmmmm
c c c c c c c 1 g c c c c c c c c c c c c c c c c 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 mmmmmmmmmmmmmmmmmmmmmmmm
0
15|16
31|32
47|48
63
64 bits

Figure 3. Expansion of a 48-bit MAC Address into an EUI-64 Interface Identifier

The g bit denotes the local/global bit and it is set to zero for 48-bit MAC-based identifiers. The c and m bits are for the company ID and manufacturer-selected extension identifier, respectively. Hence, we will get the following kind of expansions:

```
0:3:ba:6:14:66 → 203:baff:fe06:1466
0:80:c7:54:43:4d → 280:c7ff:fe54:434d
```

IPv6 does not have the Address Resolution Protocol[7] to convert IPv6 addresses to MAC addresses. Instead, this function has been transferred to the ICMPv6 Neighbor Discovery protocol, which provides generic mechanisms to discover the MAC addresses of the other hosts. Neighbor Solicitation is either multicast or unicast, and Neighbor Advertisement (the reply) is always unicast.

One may well use the same interface identifier on multiple interfaces as long as the subnet prefix is different. RFC-2373 states the following:

> Note that the use of the same interface identifier on multiple interfaces of a single node does not affect the interface identifier's global uniqueness or each IPv6 address's global uniqueness created using that interface identifier.

For example, Sun hardware assigns the same MAC address to all Ethernet interfaces. This is perfectly legal as long as the interface identifier is unique to each link (e.g., Ethernet segment) and the complete IPv6 address (64 bits of the subnet prefix and 64 bits of the interface identifier) is globally unique.

UNSPECIFIED ADDRESS

The first unicast address type you will encounter is the unspecified address:

```
0:0:0:0:0:0:0:0
```

or, in the compressed form:

```
::
```

The unspecified address is only used when an interface does not have a unicast address assigned to it yet. It can never be assigned to a node or used as a destination address.

LOOPBACK ADDRESS

The loopback address is the equivalent of 127.0.0.1 and cannot be used on a physical interface. The IPv6 loopback address consists of 127 binary zeros ending with binary one:

```
0:0:0:0:0:0:0:1
```

or in the compressed form:

```
::1
```

As with IPv4, the loopback address is always associated within a single node. You cannot use it as the source or the destination addresses for packets you are sending over a link to another node.

LINK-LOCAL ADDRESS

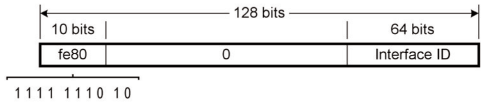


Figure 4. Link-local Address

You can use this address type (Figure 4) on a subnet (e.g., Ethernet segment) the host is connected to. The link-local address is configured automatically for each IPv6-capable interface. Since there is only the prefix, fe80::/10, but no subnet identifier, these addresses cannot be routed.

The link-local address space is roughly equivalent to the IPv4 169.254.0.0/16 address space[8] that DHCP auto-configured hosts are expected to use if no DHCP server is present. It is pointless to attempt to route link-local addresses, since there will be other link-local addresses on the destination network and the destination host will have no idea which is the correct host.

On a Solaris system, your interface list will look like the following (after the interfaces have been configured for IPv6 but the local router has not handed out the subnet prefix yet. . . or there may not be a local router at all):

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,
                MULTICAST,IPv6> mtu 8252 index 1
     inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6>
                mtu 1500 index 2
     ether 0:3:ba:e:6a:6a
     inet6 fe80::203:baff:fe0e:6a6a/10
```

The bottom line is that link-local addresses are instant: You don't have to ask for an IPv6 address to be able to communicate with the other IPv6 hosts on the same segment. You only need to hook up to the network and the other hosts are immediately reachable, without any waiting periods for a DHCP server that isn't there. Finally, since MAC addresses are globally unique, there can be up to 2^{64} nodes on a single IPv6 LAN segment.

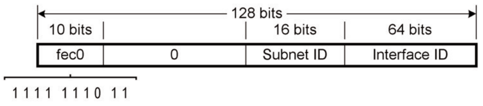


Figure 5. Site-local Address

SITE-LOCAL ADDRESS

The site-local unicast address (Figure 5) is the IPv6 equivalent of the 10.0.0.0/8-network.[9] Site-local address is for organizations that want to set up private routed IPv6 internetworks without Internet connectivity.

The site-local address prefix is fec0::/10. There are only 16 bits for the subnet identifier, which is equal to the two middle octets of the 10.0.0.0/8 network, i.e., 2^{16} subnets. Since the subnet and the interface identifier are separate entities, it is not possible to increase the number of subnets at the expense of hosts on each subnet. The interface identifier is a 64-bit fixed-length field that cannot be shortened. On the other hand, with IPv6 one does not have to put any all-zeros and all-ones addresses aside for network numbers and broadcast addresses, respectively, since each subnet is denoted by its own subnet identifier and there is no broadcast in IPv6.

Today, it seems that a high fraction of enterprise networks are using RFC-1918[9] address space and resort to Network Address Translation (NAT) to communicate with the rest of the world. To make things worse, many of them confuse NAT with network security even though the two main goals of NAT are:

- To extend the registered official address space one has received from the ISP, e.g., five official addresses on the outside and a mixture of private networks on the inside.
- To get at least some independence from one's Internet Service Provider. It is difficult enough to get official IPv4 network numbers, but renumbering all the IPv4 networks and hosts is a nightmare if the ISP changes.

It remains to be seen whether the IPv6 address space is large enough to convince enterprise network administrators to put site-local addresses aside and use registered (i.e., aggregateable global unicast) IPv6 addresses instead. Moreover, IPv6 networks are much easier to renumber since all nodes get the network prefix(es) automatically from the nearest router; there can be multiple prefixes on an interface simultaneously, and if things start to go wrong, one can always get to a host via a link-local address. If these arguments are not enough, there will be a market for IPv6 network address translation.

As your IPv6 interface starts up, it will receive its network prefix from its default gateway:

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,
                    MULTICAST,IPv6> mtu 8252 index 1
     inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6>
                    mtu 1500 index 2
     ether 0:3:ba:e:6a:6a
     inet6 fe80::203:baff:fe0e:6a6a/10
eri0:1: flags=2080841<UP,RUNNING,MULTICAST,
                    ADDRCONF,IPv6> mtu 1500 index 2
     inet6 fec0::a801:203:baff:fe0e:6a6a/64
```

Depending on your IPv6 implementation, additional IPv6 addresses may show up as subinterfaces (Solaris) or aliases (KAME[10] stack on BSD). This is merely a matter of representation, since the two forms share the same functionality.

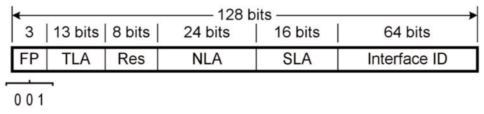


Figure 6. Aggregatable Global Unicast Address

AGGREGATABLE GLOBAL UNICAST ADDRESS

You need the aggregateable global unicast address (Figure 6) to run IPv6 on the Internet, that is, if your ISP is connected to 6bone (*http://www.6bone.net/*) and they have registered a prefix for themselves. The address type is specified in RFC-2374.[11]

The Format Prefix (FP) is always binary 001 for this type of an address. Since the length of FP is three bits and the Top-Level Aggregation Identifier (TLA) is 13 bits, the TLA is actually 16 bits, with the FP restricting the values TLA may have. Hence, the possible values for the TLAs are 2000::/16 through 3fff::/16.

The Top-Level Aggregation Identifier is the highest level in the routing hierarchy. A large ISP could have a TLA, but currently there is no TLA for a single organization. Instead, three types of TLAs have been allocated:

- Regional Internet Registry allocated sub-TLAs (prefix 2001::/16) a.k.a. Production sub-TLAs.
- 6to4 TLAs (prefix 2002::/16) to route IPv6 traffic over the IPv4 Internet.
- 6bone pseudo-TLAs (pTLAs, prefix 3ffe::/16).

The Reserved field is for future use and is always zero. However, in the following section we will discuss the previously mentioned TLA types more closely, and all of them use the address space put aside in the Reserved field.

The Next-Level Aggregation Identifier is used by organizations that have received a TLA to create an addressing hierarchy. Instead of using the space of 24 bits as a flat index, RFC-2374 recommends putting aside a number of bits as a selector and reserving the remaining space for each site the organization has to support (Figure 7).

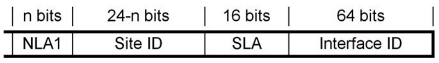


Figure 7. NLA and Site Identifiers

The organization may also choose to use the site identifier space to create an NLA hierarchy and manage routing tables more efficiently (Figure 8). This is not mandated in RFC-2374, but the recipients of TLAs are advised to set up their addressing in a hierarchical manner.

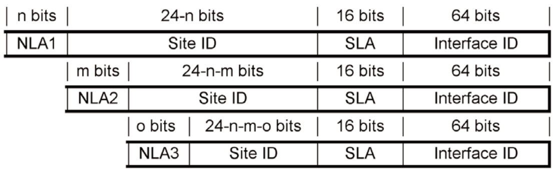


Figure 8. NLA Hierarchy

The bottom line is that 24 bits of NLA is a lot of real estate for anybody. The designers of IPv4 did not want to use their 24 bits of network space as a flat index but created the structure of Class A, B, and C networks – and added Classless Inter-Domain Routing later – to be able to control address allocation. Similarly, it does make sense to set up an NLA hierarchy since otherwise one might end up with 2^{24} entries in the routing table.

The Site-Level Aggregation Identifier is the registered equivalent of the subnet identifier in the site-local address. It can accommodate a total of 2^{16}, or 65,536, networks with up to 2^{64} nodes on each. However, it is the responsibility of each administering organization to maintain its address space reasonably well, and the current state-of-the-art is to set up an addressing hierarchy (Figure 9).

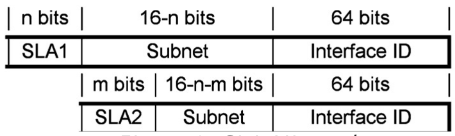


Figure 9. SLA Hierarchy

There is an interesting consequence of using 16 bits both for the SLA and the subnet identifier. Even if the organization decides to start with site-local addresses and upgrade to aggregateable global unicast addresses later, there is an upgrade path. Both site-local addresses and the various types of aggregateable global unicast addresses have 16 bits assigned for the local network identifier. Hence, one will not have to change the local addressing structure to get or change an ISP, since this information remains untouched in the process.

While your host auto-configures its IPv6 interfaces, you will see new entries in the interface list. Note how it is perfectly legal to use the same network value (0xa801) both in the subnet identifier and SLA for the interfaces eri0:1 and eri0:2, respectively. Interface renumbering has never been this easy.

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,
                MULTICAST,IPv6> mtu 8252 index 1
     inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6>
                mtu 1500 index 2
     ether 0:3:ba:e:6a:6a
     inet6 fe80::203:baff:fe0e:6a6a/10
eri0:1: flags=2080841<UP,RUNNING,MULTICAST,
                ADDRCONF,IPv6> mtu 1500 index 2
     inet6 fec0::a801:203:baff:fe0e:6a6a/64
eri0:2: flags=2080841<UP,RUNNING,MULTICAST,
                ADDRCONF,IPv6> mtu 1500 index 2
     inet6 2001:11f8:5ef9:a801:203:baff:fe0e:6a6a/64
```

OTHER AGGREGATABLE GLOBAL UNICAST ADDRESSES

In the following sections we will discuss other aggregateable global unicast address types. It would have been too easy to have only one address structure and, therefore, IETF has invented three of them, depending on the address prefix one is using.

TLA 1

Production sub-TLAs (prefix 2001::/16) are meant to be used on native IPv6 infrastructure. This is also known as the TLA 1, and the initial prefix assignments can be found in RFC 2928.[12] To make things more complicated, the TLA 1 addresses have a structure that differs from the one discussed before. (Figure 10)

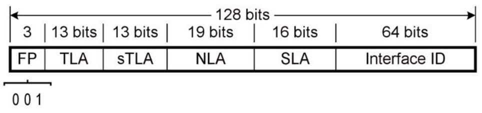


Figure 10. TLA 1 Address

All sub-TLA addresses share the same TLA prefix, 2001::/16, but the sTLA portion of the address is allocated by the Regional Internet Registries, i.e., APNIC, ARIN, and RIPE. As of September 11, 2002, the RIRs had allocated 219 sTLA prefixes.

The length of the sTLA is 13 bits, i.e., the total prefix length FP + TLA + sTLA should be 29 bits. However, the Global IP Allocation List[13] at RIPE shows prefix lengths of 32 and 35 bits:

WIDE-JP-19990813	2001:0200::/35
EU-UUNET-19990810	2001:0600::/35
UK-JANET-19991019	2001:0630::/32

Obviously, this extends the sub-TLA to the Next-Level Aggregation Identifier and leaves 16 bits or 13 bits (/32 and /35 prefixes, respectively) for the NLA. On the other hand, even with 13 bits for the NLA and 16 bits for the Site-Level Aggregation (SLA) Identifier one can design quite a respectable network.

6to4 TLA

6to4 TLAs (prefix 2002::/16) are usedto route IPv6 traffic over the IPv4 Internet and even over IPv4 dialups, if necessary. This TLA and the address format are discussed in Carpenter et al.[14] and Carpenter and Moore.[15] The address format is shown in Figure 11.

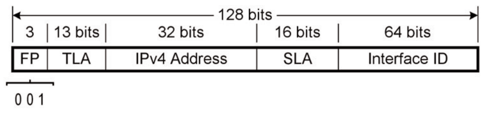


Figure 11. 6to4 Address

6to4 is an alternative vehicle if you want to run IPv6 but don't have access to a 6bone gateway. In this case, your IPv6 address is constructed from your registered IPv4 address. Of course, if you are dialing in to an ISP or your cable modem/DSL provider assigns your address via DHCP, your IPv4 address won't be static and you cannot run any persistent services.

6bone TLA

6bone (prefix 3ffe::/16) is a global IPv6 testbed on the Internet. Originally 6bone links were IPv6-over-IPv4 tunnels, but the infrastructure is gradually moving toward native IPv6 links and routers. On the practical side of things, one is most likely to get a 6bone address if one wants to run IPv6 outside a single LAN or an organization. Since the ISPs and other organizations that have registered a TLA 1 address may not be able to deliver an IPv6 service to end customers for a long time, your most viable route to the IPv6 world is to use 6bone addresses and IPv6-over-IPv4 encapsulation.

The 6bone pTLA address is shown in Figure 12:

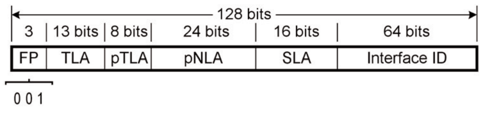


Figure 12. 6bone pTLA Address

As of September 3, 2002, there were 132 6bone prefixes allocated. Unlike RIR-allocated sub-TLAs, there were /24, /28, and /32 prefixes:

ROOT66/US-CA	3ffe:0000::/24
TRUMPET/AU	3ffe:8000::/28
TELEPAC/PT	3ffe:4000::/32

If you don't have any 6bone gateway you can access, probably the easiest way to get connected is to use the TSP client provided by Freenet6 (*http://www.freenet6.net/*).

Multicast Addresses

While multicast came to IPv4 only later, in the form of the D address class,[16] it has been a critical part of IPv6 from the very beginning. The major advantage that multicast has over broadcast is better control of propagation, since an application will not have to reach every station on the network but only those nodes that actually need the data. The format of the multicast address is in Figure 13:

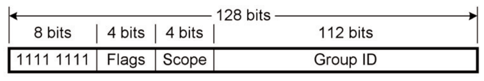


Figure 13. Multicast Address

The prefix for multicast is all-ones (i.e., binary 1111 1111 or ff::/8). The next four bits are the flags (Figure 14), but only the last bit is in use. The T bit must be 0 for permanently assigned multicast addresses (visit *http://www.iana.org/* for more information) and 1 for non-permanent (i.e., transient) addresses. The first three bits are reserved for future use and must be set to zero.

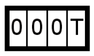


Figure 14. Multicast Flags

While IPv4 multicast has few propagation-controlling mechanisms other than time to live, IPv6 multicast has a clearly defined scope:

Value	Scope
0	Reserved
1	Node-local scope
2	Link-local scope
3	—
4	—
5	Site-local scope
6	—
7	—
8	Organization-local scope
9	—
a	—
b	—
c	—
d	—
e	Global scope
f	Reserved

Table 2. Multicast Scope Values

For example, the assigned Group Identifier for Routing Information Protocol Next Generation (RIPng) routing protocol is 9 (hex) and thus all RIPng updates are sent using the multicast address ff02::9, i.e., the scope is for the local link only. It is a bit difficult to imagine any practical use for route updates beyond the local link, but an application such as Network Time Protocol could well have several scope values:

ff01::101 means all NTP servers on the same node as the sender.
ff02::101 means all NTP servers on the same link (e.g., Ethernet LAN) as the sender.
ff05::101 means all NTP servers in the same site as the sender.
ff08::101 means all NTP servers in the same organization as the sender.
ff0e::101 means all NTP servers in the Internet.

Note that transient multicast addresses are relevant only within the given scope. For example, the transient, site-local address ff15::101 is a completely different entity from the transient, link-local address ff12::101 or the permanent address ff05::101. Transient addresses are disposed of at the end of their use.

Since multicast has a critical role in IPv6 and all IPv6 hosts and nodes must be multicast-capable, it is likely that the emergence of IPv6 will also enable large-scale deployment of multicast gaming and multicast audio and video over the Internet.

Auto-configuration

Address auto-configuration is one of the many areas where IPv6 makes sense to an end user. IPv6 defines two modes of auto-configuration: stateless and stateful. All IPv6 hosts are capable of stateless auto-configuration,[17] in which they generate their link-local address from the MAC addresses, possibly using some external source (the nearest router) to generate the site-local and aggregate-able global unicast address. At the end of the auto-configuration process there is duplicate detection to verify uniqueness of the addresses and to prevent address collisions with the other hosts on the same link.

The most common example of stateful auto-configuration is a DHCP server. While stateful auto-configuration is perfect for setting up the addresses and the default route, it does not cover things like DNS resolution, path to the boot image, timeservers, and other information one might need for host configuration.

In practice, one may use both stateless and stateful auto-configuration to set up a host. Stateless auto-configuration is used for the initial configuration, and the rest of the parameters are obtained through a stateful process. Consequently, the auto-configuration server does not necessarily have to share the same link with the host, although this could be desirable for administrative reasons. In other words, if things start going wrong the administrator does not have to troubleshoot both the network and the server. Moreover, in large organizations these are two different groups that may not even want to talk to each other, and having a critical organization-wide resource behind multiple routers could become fertile ground for finger-pointing and other corporate trouble.

SOLARIS HOST

The information is this section is based on the Solaris documentation[18] and Zilbauer.[19] You can set up Solaris 8 or Solaris 9 in just a few steps:

1. Find out the device name for your primary Ethernet interface. There are a few ways to do this but let us use ls:

```
# ls -l /etc/hostname*
-rw-r--r--    1 root   root      23 Jun 15 14:36 /etc/hostname.eri0
```

2. Create an empty file /etc/hostname6.interface and reboot:

```
# touch /etc/hostname6.eri0
# ls -l /etc/hostname*
-rw-r--r--    1 root   root      23 Jun 15 14:36 /etc/hostname.eri0
-rw-r--r--    1 root   other      0 Sep 12 13:37 /etc/hostname6.eri0
# reboot
```

   Rebooting is necessary due to the way Solaris initializes interfaces for IPv4 and IPv6.

3. After the system has come back up you will see IPv6 active on the loopback and the Ethernet interfaces:

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 1
    inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6> mtu 1500 index 2
    ether 0:3:ba:e:6a:6a
    inet6 fe80::203:baff:fe0e:6a6a/10
```

4. Edit /etc/inet/ipnodes[20] to add the addresses to the host table. For some reason Sun didn't want to continue to use /etc/hosts but created a completely new hosts file instead. Interestingly, the syntax of the file has not changed from /etc/hosts. This is a perfectly valid reason to start using DNS with IPv6 as soon as possible.

   ```
   # vi /etc/inet/ipnodes
   "/etc/inet/ipnodes" [Read only] 17 lines, 500 characters
   :a
   fe80::203:baff:fe0e:6a6a  mysun.v6.mydomain.org mysun.v6 ll-mysun
   .
   :wq!
   "/etc/inet/ipnodes" 18 lines, 522 characters
   ```

   Note how we used the .v6 subdomain for the IPv6 addresses. Bucklin and Sekiya[21] make an excellent argument for separating IPv6 addresses from the IPv4 ones. Most importantly, there are quite a few resolvers in use that do not understand the AAAA record and may fail when they get such a record back. On the other hand, since it is uncertain how IPv6 will be rolled out to organizations, keeping the IPv6 and IPv4 namespaces separate at this stage seems a practical solution.

   Naming conventions for IPv6 hosts can be a bit tricky. While IPv4 usually defines only one IP address for an interface, it is common to have at least two IPv6 addresses for an interface: one link-local and one routable. Hence, to distinguish between these two we will prefix the hostname with ll- (e.g., ll-mysun) for the link-local address while the fully qualified domain name points to the routable address. The interface name is another possible prefix.

5. We also have to edit /etc/nsswitch.conf to enable host lookups from DNS as well as /etc/inet/ipnodes:

   ```
   # ex /etc/nsswitch.conf
   "/etc/nsswitch.conf" 26 lines, 690 characters
   :/^ipnodes
   ipnodes:        files
   :s/files/files dns/
   ipnodes:        files dns
   :wq!
   "/etc/nsswitch.conf" 26 lines, 694 characters
   ```

6. As we discussed in the section on Link-local Address, you can immediately communicate with other IPv6 hosts on the local LAN. As always, pinging them is a good way to start, and if ping works, Telnet will probably too:

   ```
   # ping -A inet6 fe80::2e0:98ff:fe83:48d0
   fe80::2e0:98ff:fe83:48d0 is alive
   ```

Now you are all set. Auto-configuration is about the only reasonable way to set up an IPv6 host, since it guarantees consistent setup across all your hosts on the network. Of course, without a router there will not be anyone to hand out a site-local prefix or an aggregateable global unicast prefix, and you will have to use link-local addresses only.

FREEBSD HOST

FreeBSD and other BSD variants running the KAME stack are easy to configure for IPv6. The GENERIC kernel in FreeBSD-4.6 has IPv6 enabled by default, and you only have to add the following line to your /etc/rc.conf and reboot:

```
ipv6_enable="YES"
```

Your Ethernet interface is probably set up for IPv6 anyway, but this entry will guarantee that the host will also request the prefix from the nearest router and sets up its routing table accordingly.

1. Check that there is ipv6_enable entry in /etc/rc.conf and /etc/defaults/rc.conf:

   ```
   # grep ipv6_enable /etc/rc.conf
   # grep ipv6_enable /etc/defaults/rc.conf
   ipv6_enable="NO"        # Set to YES to set up for IPv6.
   #
   ```

   In this case the default setting is "NO" and we have to override it in /etc/rc.conf:

2. Add two entries, ipv6_enable and ipv6_network_interfaces, to the end of /etc/rc.conf:

   ```
   # ex /etc/rc.conf
   "/etc/rc.conf " 22 lines, 625 characters
   :$
   :a

   # IPv6 configuration settings
   ipv6_enable="YES"
   ipv6_network_interfaces="auto"
   .
   :wq
   "/etc/rc.conf " 24 lines, 668 characters
   ```

3. Reboot.

   ```
   # shutdown -r now
   ```

   When the host boots up it will obtain the prefixes and routing information from the nearest router, if any. If no router is live, it will use its link-local address until a router becomes available.

4. Log in and run ifconfig(8) after the system has rebooted:

   ```
   # ifconfig -a
   lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
       inet6 ::1 prefixlen 128
       inet6 fe80::1%lo0 prefixlen 64 scopeid 0x2
       inet 127.0.0.1 netmask 0xff000000
   ed1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
       inet6 fe80::2e0:98ff:fe83:48d0%ed1 prefixlen 64 scopeid 0x3
       inet6 fec0::a801:2e0:98ff:fe83:48d0 prefixlen 64 autoconf
       inet 192.168.35.51 netmask 0xffffff00 broadcast 192.168.35.255
       ether 00:e0:98:83:48:d0
       media: Ethernet 10baseT/UTP
       status: active
   ```

Note that the KAME stack has a funny habit of appending the interface name at the end of the link-local address. Hence, to ping the Solaris host we just configured we have to issue the following command (unlike Solaris, FreeBSD has ping and ping6 for IPv4 and IPv6, respectively):

```
# ping6 fe80::203:baff:fe0e:6a6a%ed1
PING6(56=40+8+8 bytes) fe80::2e0:98ff:fe83:48d0%ed1 --> fe80::203:baff:fe0e:6a6a%ed1
16 bytes from fe80::203:baff:fe0e:6a6a%ed1, icmp_seq=0 hlim=64 time=0.12 ms
16 bytes from fe80::203:baff:fe0e:6a6a%ed1, icmp_seq=1 hlim=64 time=0.121 ms
16 bytes from fe80::203:baff:fe0e:6a6a%ed1, icmp_seq=2 hlim=64 time=0.134 ms
^C
--- fe80::203:baff:fe0e:6a6a%ed1 ping6 statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 0.120/0.125/0.134/0.006 ms
```

The link-local address notation adopted by KAME does have its advantages when you have a multi-homed host, especially if you have a Sun box running some variant of BSD. Because all Sun Ethernet interfaces share the same MAC address, the link-local address becomes ambiguous – one does not know which interface to use to reach the destination. For example, the following link-local address could refer to any interface on a multi-homed Sun box:

```
fe80::a:a20:22ff:fe6d:6e
```

In contrast, the KAME notation explicitly points to the right interface:

```
fe80::a:a20:22ff:fe6d:6e%le0
fe80::a:a20:22ff:fe6d:6e%le1
```

Of course, the discussion on interface identifiers in the link-local address has meaning only on the host one has the KAME stack on. In other words, do not add the interface extension to the IPv6 address if you are pinging a FreeBSD host from Solaris:

```
# ping -A inet6 fe80::2e0:98ff:fe83:48d0%ed1          ← This will not work
ping: unknown host fe80::2e0:98ff:fe83:48d0%ed1

# ping -A inet6 fe80::2e0:98ff:fe83:48d0              ← This works!
fe80::2e0:98ff:fe83:48d0 is alive
```

SOLARIS ROUTER

If you want to connect your IPv6 LAN to the rest of the world you will need a router. Cisco, Nortel, and Juniper, to name a few vendors, all have IPv6 support in at least some of their products. However, we will choose the poor man's alternative and use a dual-homed Solaris box instead.

The basic configuration of a Solaris router does not differ from setting up a Solaris node. Since we have two Ethernet interfaces, we have to create a hostname6.*interface* file for them both.

1. Find out your interface names and create empty /etc/hostname6.interface for all physical devices. If both interfaces are on the motherboard (as is the case with Netra X1) or you have a quad Ethernet card, figuring out the second interface is simple.

```
# ls -l /etc/hostname*
-rw-r--r--     1 root   root     23 Jun 15 14:36 /etc/hostname.dmfe0
```

Since Netra X1 has two Ethernet interfaces and the primary is dmfe0, we can make an educated guess that the secondary interface is dmfe1. Now we only have to create the hostname6 files.

```
# touch /etc/hostname6.dmfe0 /etc/hostname6.dmfe1
# ls -l /etc/hostname*
-rw-r--r--     1 root   root     23 Jun 15 14:36 /etc/hostname.dmfe0
-rw-r--r--     1 root   other    0 Sep 12 13:37 /etc/hostname6.dmfe0
-rw-r--r--     1 root   othe     0 Sep 12 13:37 /etc/hostname6.dmfe1
```

Normally we would reboot the box right after creating hostname6.*interface*, but this box is going to be a router. Hence, change the default directory to /etc/inet and create ndpd.conf to set up prefixes for Neighbor Discovery.

2. Edit /etc/inet/ndpd.conf to set up prefixes for each IPv6 interface. To keep things simple we are going to use site-local addresses in our example. The following sample configuration has been adapted from Solaris documentation:

```
# cd /etc/inet
# ex ndpd.conf
"ndpd.conf" [New file]
:a
ifdefault AdvReachableTime 30000 AdvReTransTimer 2000
prefixdefault AdvValidLifetime 240m AdvPreferredLifetime 120m

# 0xa801 = 168.1
if dmfe0 AdvSendAdvertisements 1
```

```
prefix fec0:0:0:a801::/64 dmfe0

# 0xa802 = 168.2
if dmfe1 AdvSendAdvertisements 1
prefix fec0:0:0:a802::/64 dmfe1
.
:wq
"ndpd.conf" 8 lines, 248 characters
# reboot
```

Now we are all set for reboot. When the system boots up it will create the IPv6 interfaces and auto-configure them. Consequently, the in.ndpd auto-configuration daemon will start up and assign the prefixes for the interfaces. It will also hand out the prefixes for all the stations on the local Ethernet. At the end of the process, the RIPng routing daemon will start up to advertise and listen for route updates on both sides of the router.

3. Log in and check the IPv6 interfaces list after the host has booted up. You should see the link-local interfaces the host has configured for itself and the site-local prefix it has received from the router:

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 1
    inet6 ::1/128
dmfe0: flags=2100841<UP,RUNNING,MULTICAST,ROUTER,IPv6> mtu 1500 index 2
    ether 0:3:ba:6:14:66
    inet6 fe80::203:baff:fe06:1466/10
dmfe0:1: flags=2180841<UP,RUNNING,MULTICAST,ADDRCONF,ROUTER,IPv6> mtu 1500 index 2
    inet6 fec0::a801:203:baff:fe06:1466/64
dmfe1: flags=2100841<UP,RUNNING,MULTICAST,ROUTER,IPv6> mtu 1500 index 3
    ether 0:3:ba:6:14:66
    inet6 fe80::203:baff:fe06:1466/10
dmfe1:1: flags=2180841<UP,RUNNING,MULTICAST,ADDRCONF,ROUTER,IPv6> mtu 1500 index 3
        inet6 fec0::a802:203:baff:fe06:1466/64
```

4. Before we continue any further, it is a good practice to record the IP addresses in /etc/inet/ipnodes and enable IPv6 DNS lookups in /etc/nsswitch.conf:

```
# vi /etc/inet/ipnodes
"/etc/inet/ipnodes" [Read only] 17 lines, 500 characters
:a
fe80::203:baff:fe06:1466            sun-gw.v6.mydomain.org sun-gw.v6 ll-sun-gw
fec0::a801:203:baff:fe06:1466       dmfe0-sun-gw.v6.mydomain.org dmfe0-sun-gw
fec0::a802:203:baff:fe06:1466       dmfe1-sun-gw.v6.mydomain.org dmfe1-sun-gw
.
:wq!
"/etc/inet/ipnodes" 18 lines, 522 characters
```

It is a bit difficult to find reasonable IPv6 naming conventions on Sun hardware, since Sun insists on using the same MAC address on every interface. One way around this is to use the ll- prefix for all the link-local addresses and the interface name for all routable addresses. It is not perfect but, after all, /etc/inet/ipnodes is like /etc/hosts: you use the file to get the most critical interface names (i.e., yours) and obtain everything else from the DNS. In other words, this minimal /etc/inet/ipnodes would be as follows:

```
::1                                 localhost.mydomain.org localhost
fe80::203:baff:fe06:1466            sun-gw.v6.mydomain.org sun-gw.v6 ll-sun-gw
```

For maximum flexibility you may not even want to have anything other than the link-local address in /etc/inet/ipnodes , since the link-local address will always be associated with the device. Otherwise, all routable addresses, be those site-local or aggregateable global unicast ones, become elusive. If the host knows only its link-local address and gets everything else from the DNS, there is one issue less to worry about when you eventually have to renumber the network.

Finally, the -gw suffix to the hostname is in accordance with the ipnodes(4) manual page, which, in turn, refers to RFC-952.[22]

5. We also have to edit /etc/nsswitch.conf to enable host lookups from DNS as well as /etc/inet/ipnodes:

```
# ex /etc/nsswitch.conf
"/etc/nsswitch.conf" 26 lines, 690 characters
:/^ipnodes
ipnodes:    files
:s/files/files dns/
ipnodes:    files dns
:wq!
"/etc/nsswitch.conf" 26 lines, 694 characters
```

This is everything you have to do to get started. The existing hosts on both sides will learn of the gateway when it boots up and will adjust the interface and routing tables accordingly. For example, there is a new IPv6 address for the Solaris workstation we configured earlier:

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 1
      inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6> mtu 1500 index 2
      ether 0:3:ba:e:6a:6a
      inet6 fe80::203:baff:fe0e:6a6a/10
eri0:1: flags=2080841<UP,RUNNING,MULTICAST,ADDRCONF,IPv6> mtu 1500 index 2
      inet6 fec0::a801:203:baff:fe0e:6a6a/64
```

Similarly, the routing table has changed to indicate a new default gateway:

```
Routing Table: IPv6
  Destination/Mask   Gateway                                   Flags   Ref    Use     If
------------------------------ ---------------------------------------------- ------- ----- ------ -------
fec0:0:0:a801::/64   fec0::a801:203:baff:fe0e:6a6a             U       1      2       eri0:1
fe80::/10            fe80::203:baff:fe0e:6a6a                  U       1      1       eri0
ff00::/8             fe80::203:baff:fe0e:6a6a                  U       1      0       eri0
default              fe80::203:baff:fe06:1466                  UG      1      0       eri0
::1                  ::1                                       UH      1      0       lo0
```

And this is how it looks on FreeBSD:

```
Routing Tables

Internet6:
Destination                     Gateway                          Flags      Netif   Expire
::/96                           ::1                              UGRSc      lo0     =>
default                         fe80::203:baff:fe06:1466%ed1     UGc        ed1
::1                             ::1                              UH         lo0
::ffff:0.0.0.0/96               ::1                              UGRSc      lo0
fe80::/10                       ::1                              UGRSc      lo0
fe80::%lo0/64                   fe80::1%lo0                      Uc         lo0
fe80::1%lo0                     link#2                           UHL        lo0
fe80::%ed1/64                   link#3                           UC         ed1
fe80::203:baff:fe06:1466%ed1    00:03:ba:06:14:66                UHLW       ed1
fe80::2e0:98ff:fe83:48d0%ed1    00:e0:98:83:48:d0                UHL        lo0
fec0:0:0:a823::/64              link#3                           UC         ed1
fec0::a801:203:baff:fe06:1466   00:03:ba:06:14:66                UHLW       ed1
fec0::a801:2e0:98ff:fe83:48d0   00:e0:98:83:48:d0                UHL        lo0
ff01::/32                       ::1                              U          lo0
ff02::/16                       ::1                              UGRS       lo0
ff02::%lo0/32                   ::1                              UC         lo0
ff02::%ed1/32                   link#3                           UC         ed1
```

If sometime in the future you decide to open up your network to the world and get connected to 6bone, you only have to advertise the new 6bone TLA or production sub-TLA prefix and the hosts will pick it up instantly.

MANUAL CONFIGURATION

It is also possible to configure the IPv6 address manually on Solaris. This makes little sense on a local LAN, but it is quite necessary, for example, when you have to define the endpoints of an IPv6 over IPv4 tunnel. However, on a local LAN this would only contribute to the administrator's headache, since you would have to manually change this configuration every time you renumber the local LAN. If you used auto-configuration instead, the nearest router would take care of handing out the prefixes.

To summarize, there are situations when you have to configure the IP address manually, but the local LAN would not be involved in any of those. However, for the sake of simplicity the following example will show manual configuration of the primary Ethernet interface:

1. Check the interface name for your primary Ethernet interface:

```
# ls -l /etc/hostname*
-rw-r--r-- 1 root    root   23 Jun 15 14:36 /etc/
                                        hostname.eri0
```

2. Enter the prefix parameters into /etc/hostname6.eri0:

```
# cat > /etc/hostname.eri0
addif fec0::2001:203:baff:fe0e:6a6a up
Ctrl-D
#
```

3. Reboot.

HOW DOES IT WORK?

Since RFC-2462 specifies the process for stateless address auto-configuration, we will only examine the process as it appears on the network. There are some variations to the process, depending on your particular stack implementation, but Figure 15 displays the operation at a generic level:

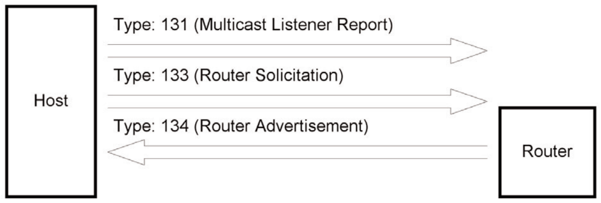

Host
Type: 131 (Multicast Listener Report)
Type: 133 (Router Solicitation)
Type: 134 (Router Advertisement)
Router

Figure 15. Stateless Address Auto-configuration

The first step in auto-configuration is for the host to generate the link-local address for itself. If the host is connected to a LAN and therefore has a MAC address, this is a fairly straightforward procedure:

```
0:3:ba:e:6a:6a → fe80::203:baff:fe0e:6a6a
```

Although MAC addresses are supposed to be globally unique, this is not always the case. Some vendors have used unregistered or duplicate MAC addresses in their low-end Ethernet interfaces and there is a possibility that one of these devices is on the same LAN segment with the auto-configuring host.

The duplicate detection algorithm is relatively straightforward. After the host has configured a link-local address for itself, it will join the All-Nodes multicast group by sending a Multicast Listener Report and announcing the multicast address it is listening to:

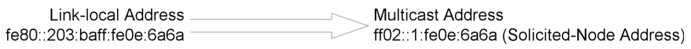


Figure 16. Multicast Listener Report

If there is another host listening at the same address, it will notify the host and give its link-local address. If there is a collision, the host may modify its address and try again, or stop the process altogether and alert the operator to intervene. Duplicate address detection is not perfect; it cannot detect offline hosts or the reply may be lost due to a collision. Duplicate detection still is a cheap sanity check to prevent the most obvious address problems.

After the host has verified that it has a more or less unique link-local address, it continues the auto-configuration process by sending a Router Solicitation to the All-Routers multicast address:

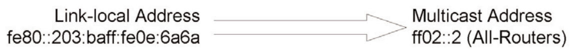


Figure 17. Router Solicitation

The router on the local segment will reply with a Router Advertisement sent to the All-Nodes multicast address (Figure 18). The Router Advertisement will contain the prefixes for the network and the lifetime of the prefix. Note that it makes perfect sense to use the All-Nodes address for the Router Advertisement. There could be multiple hosts in the midst of auto-configuration, in which case it is cheaper to use multicast as opposed to replying with a unicast to each and every host that sent a Router Solicitation.

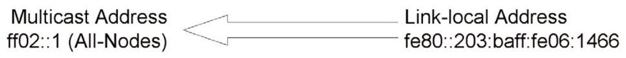


Figure 18. Router Advertisement

Since high availability seems to be the most repeated mantra in quite a few TCP/IP networks, there is a tendency to have a router pair on critical LAN (i.e., Ethernet) segments. Since the routers are supposed to use HSRP or VRRP to monitor each other's status, only the master router can reply to the Router Advertisement. Depending on the configuration, the master may use either its own interface address or the high availability virtual address.

A host should not know anything about routing. The idea of Router Solicitation and Router Advertisement was already in ICMPv4 but it did not gain much popularity. IPv6 takes the concept further, to the extent that one cannot have a routable IPv6 address if there is no router to hand out the prefix. Yet the hosts can communicate with each other on the local segment, using the link-local addresses.

Vendors, such as Sun Microsystems, may decide to add their own solicitation messages to auto-configuration (Figure 19). Sun Solaris registers itself to the multicast group ff02:202 a.k.a. Sun RPC PMAPPROC_CALLIT. This group appears to be for the communication of Sun RPC portmapper to locate some particular RPC service.

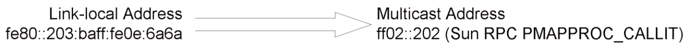


Figure 19. Sun RPC Portmapper

Stateless address auto-configuration is a great concept, but like all concepts, it can be subverted by stupid designs. For example, Sun Netra X1 has two Ethernet interfaces on the motherboard. Some designers got the bright idea of using the two interfaces on different subnets and setting up route metrics to use either side as the default gateway. IPv6 auto-configuration will not permit this, since the router sending the Router Advertisement will also become the default gateway, and having two default gateways will cause asymmetric routing. One would have to bypass auto-configuration with fixed prefixes and run RIPng on the host without packet forwarding to enable this kind of setup, thus losing the automations (and benefits of IPv6) described above.

IPv6 and DNS

The status of the DNS in the IPv6 world is still in flux. BIND 8 implemented the RFC-1886[23]-specified IPv6 DNS extensions such as the AAAA resource record and the ip6.int domain for reverse lookups. There is no true IPv6 name resolution over IPv6 with BIND 8, unless the server and the resolver have been patched for IPv6. Note that Solaris 9 and FreeBSD-4.x include BIND 8 in the distribution.

To confuse things further, RFC-2874[24] introduced a list of new DNS extensions for address aggregation and network renumbering and changed the outlook of IPv6 name resolution completely. This was not entirely a bad thing, since old A and PTR records were specified in the days of the Class A, B, and C networks. Although there was name delegation for forward maps, reverse map delegations for CIDR blocks have been more or less painful. CIDR blocks and network renumbering, the exercise every administrator will go through sooner or later, were not enough of a concern for forward maps.

RFC-1886 basically extended the old DNS concepts to the IPv6 world with little thought put into aggregateable global unicast addresses, TLAs, NLAs, or SLAs. With 16 octets of an address, CIDR blocks and renumbering became major issues, and the A6 and DNAME resource records and the ip6.arpa domain were proposed. The BIND 9 implementation prefers RFC-2874 but provides support for RFC-1886. The following quote is from the BIND 9 Administrator Reference Manual.[25]

> For forward lookups, BIND 9 supports both A6 and AAAA records. The use of AAAA records is deprecated, but it is still useful for hosts to have both AAAA and A6 records to maintain backward compatibility with installations where AAAA records are still used. In fact, the stub resolvers currently shipped with most operating systems support only AAAA lookups, because following A6 chains is much harder than doing A or AAAA lookups.
>
> For IPv6 reverse lookups, BIND 9 supports the new "bit-string" format used in the ip6.arpa domain, as well as the older, deprecated "nibble" format used in the ip6.int domain.

BIND 9 name server can listen to IPv6, enabling the users to use IPv6 for name lookups if they have the new Lightweight Resolver installed. The introduction of a new resolver is due to simultaneous IPv4 and IPv6 lookups and the added complexity of the A6 and DNAME resource records.

To summarize, DNS support for IPv6 will be at an introductory phase until BIND 9 and the Lightweight Resolver are available for all operating systems that have an IPv6 stack. Even then there will be a considerable installed base of BIND 8 sites, which will have a negative effect in the rollout of IPv6.

RFC-1886 LOOKUPS

FORWARD LOOKUPS

From the perspective of the named configuration file, IPv6 forward lookup zones are no different from the IPv4 ones. The configuration file specifies the name of the zone, and the protocol-specific details are kept in the zone file. For example, there isn't anything IPv6-specific in the following zone entry from the master configuration file, except the v6 subdomain:

```
zone "v6.mydomain.org" in {
      type master;
      file "db.v6.mydomain.org";
};
```

The AAAA resource record is the IPv6 equivalent of the IPv4 A record. The syntax of the AAAA resource record is exactly the same as the A record, except that the addresses take longer to type. For example, consider the following entries for the zone v6.mydomain.org:

```
$ORIGIN v6.mydomain.org.
localhost        86400     IN    AAAA   ::1
mysun            86400     IN    AAAA   fec0::a801:203:baff:fe0e:6a6a
dmfe0-sun-gw     86400     IN    AAAA   fec0::a801:203:baff:fe06:1466
dmfe1-sun-gw     86400     IN    AAAA   fec0::a802:203:baff:fe06:1466
```

REVERSE LOOKUPS

The reverse lookups are performed using the nibble format and the ip6.int domain. One octet or byte consists of two nibbles, e.g., decimal 192 equals 0xc0, which has nibbles 0xc and 0x0. The nibble format is the IPv6 equivalent of the traditional IPv4 in-addr.arpa representation, and the logic of the nibble format is exactly the same. Since the IPv6 addresses are hexadecimal, it makes a certain sense to reverse the order of the address and separate each nibble with a dot. This permits reverse zone delegation similar to the one in IPv4.

First we will examine the reverse map of the IPv6 localhost. The normal abbreviation for the localhost address is:

```
::1
```

This is equivalent to the expanded address:

```
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1
```

Consequently, we could have the following PTR resource record for localhost:

```
$ORIGIN 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.int.
1    86400   IN    PTR     localhost.v6.mydomain.org.
```

This translates to the following zone entry in the master configuration file:

```
zone "0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.int" in {
      type master;
      file "db..1";
};
```

The reverse zone entries for the prefixes fec0:0:0:a801::/64 and fec0:0:0:a802::/64 are not much different. Since we have to reverse the order of nibbles, we get the following PTR resource records:

```
$ORIGIN 1.0.8.a.0.0.0.c.e.f.ip6.int.
a.6.a.6.e.0.e.f.f.f.a.b.3.0.2.0          86400       IN          PTR      mysun.v6.mydomain.org.
0.d.8.4.3.8.e.f.f.f.8.9.0.e.2.0          86400       IN          PTR      mybsd.v6.mydomain.org.
6.6.4.1.6.0.e.f.f.f.a.b.3.0.2.0          86400       IN          PTR      dmfe0-sun-gw.v6.mydomain.org.

$ORIGIN 2.0.8.a.0.0.0.c.e.f.ip6.int.
6.6.4.1.6.0.e.f.f.f.a.b.3.0.2.0          86400       IN          PTR      dmfe1-sun-gw.v6.mydomain.org.
```

And these are the respective zone master configuration file entries:

```
zone "1.0.8.a.0.0.0.c.e.f.ip6.int" in {
      type master;
      file "db.fec0..a801";
};
```

```
zone "2.0.8.a.0.0.0.c.e.f.ip6.int" in {
      type master;
      file "db.fec0..a802";
};
```

The most obvious problem of the nibble format becomes very clear very quickly: The address prefixes and the interface identifiers are difficult to carry around, and each resource record takes a lot of proofreading to get it right. Reverse zone delegation may help here to limit the size of a single resource record but, on the other hand, it makes little sense to delegate to the interface identifier (e.g., the 203:baff portion of the aforementioned addresses). However, it does not help to set up a separate root zone for the fec0::/10 prefix, since one has to define fec0:0:0:a801::/64 and fec0:0:0:a802::/64 in the master configuration file and the zone files anyway.

RESOLVER

Note that we did not set up any resolver entry in the discussion on stateless address auto-configuration. This process could be automated with stateful address auto-configuration, but since DHCP6 is still in the works we have to set up our configuration file manually. Since we already made the decision to keep all IPv6 hosts in a separate .v6 subdomain and worry about migration later, we have to have both domain names in a search list. Moreover, since the BIND 8 resolver uses IPv4 only, we have to list our name server(s) as usual:

```
search mydomain.org v6.mydomain.org
nameserver 192.168.1.36
```

Since IPv4 applications are supposed to be in the majority, this setup will first look up for IPv4 names with mydomain.org. Since IPv6 applications do not care about A records, the lookup will proceed using v6.mydomain.org, hoping to find AAAA records. Telnet, FTP, and ping (on Solaris) will use IPv6 if available and revert back to IPv4 by default.

BIND 9 AND RFC-2874

FORWARD LOOKUPS

RFC-2874 introduced the new A6 resource record to eventually replace the AAAA record. It can be used just like the AAAA record, as the following example will show:

```
$ORIGIN v6.mydomain.org.
localhost        86400    IN    A6    0     ::1
mysun            86400    IN    A6    0     fec0::a801:203:baff:fe0e:6a6a
mybsd            86400    IN    A6    0     fec0::a801:2e0:98ff:fe83:48d0
dmfe0-sun-gw     86400    IN    A6    0     fec0::a801:203:baff:fe06:1466
dmfe1-sun-gw     86400    IN    A6    0     fec0::a802:203:baff:fe06:1466
```

These entries are no different from the aforementioned AAAA records. Site-local addresses gain little from the A6 records, since the network space is only 16 bits wide and you can have only so much address aggregation in two octets. On the other hand, changing the prefix for all the zones at a large site takes a considerable amount of work. To avoid this we can rewrite the zone by using all zeroes for the prefix:

```
$ORIGIN v6.mydomain.org.
mysun            86400    IN    A6    48    0:0:0:a801:203:baff:fe0e:6a6a prefix.v6.mydomain.org.
mybsd            86400    IN    A6    48    0:0:0:a801:2e0:98ff:fe83:48d0 prefix.v6.mydomain.org.
dmfe0-sun-gw     86400    IN    A6    48    0:0:0:a801:203:baff:fe06:1466 prefix.v6.mydomain.org.
dmfe1-sun-gw     86400    IN    A6    48    0:0:0:a802:203:baff:fe06:1466 prefix.v6.mydomain.org.
```

We can then specify the prefix with a separate resource record:

```
prefix           86400    IN    A6    0     fec0:0:0::
```

First, the lookup for mysun.v6.mydomain.org will get the A6 record 0:0:0:a801:203:baff:fe0e:6a6a. Since this record is incomplete, the query will follow the pointer to prefix.v6.mydomain.org and will receive the prefix fec0:0:0::. Then resolver on the querying node will construct the IPv6 address from the local part and the prefix and return fec0::a801:203:baff:fe0e:6a6a as the address. This process is known as the A6 chain. This is how it works with dig:

```
# dig @::1 mysun.v6.mydomain.org. A6
; <<>> DiG 9.2.1 <<>> @::1 mysun.ipv6.mydomain.org. A6
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13152
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;mysun.ipv6.mydomain.org.          IN    A6

;; ANSWER SECTION:
mysun.ipv6.mydomain.org. 86400  IN    A6    64 ::203:baff:fe0e:6a6a prefix.ipv6.mydomain.org.

;; AUTHORITY SECTION:
ipv6.mydomain.org. 86400           IN    NS    localhost.ipv6.mydomain.org.

;; ADDITIONAL SECTION:
prefix.ipv6.mydomain.org. 86400  IN    A6    0 fec0:0:0:a801::

;; Query time: 1 msec
;; SERVER: ::1#53(::1)
;; WHEN: Thu Sep 19 20:37:03 2002
;; MSG SIZE  rcvd: 146
```

There is no predefined limit to the number of hops in the A6 chain. However, since it is up to the querying application – dig in the example above – to construct the A6 chain, it makes sense to keep the chain as short as possible. If you use other tools, such as nslookup or host, you have to follow the chains manually.

REVERSE LOOKUPS

Nibble format may be rather error-prone since if something is difficult to read, it is also probably difficult to get right. The other improvement of RFC-2874 is the bitstring format for reverse zones. Simply put, the reverse zone example above looks like the following when using bitstrings:

```
$ORIGIN \[x0000000000000000000000000000000/124].ip6.arpa.
\[x1/4]                         86400   IN     PTR      localhost.v6.mydomain.org.

$ORIGIN \[xfec000a801/64].ip6.arpa.
\[x0203bafffe0e6a6a/64]    86400   IN     PTR      mysun.v6.mydomain.org.
\[x02e098fffe8348d0/64]    86400   IN     PTR      mybsd.v6.mydomain.org.
\[x0203bafffe061466/64]    86400   IN     PTR      dmfe0-sun-gw.v6.mydomain.org.

$ORIGIN \[xfec000a801/64].ip6.arpa.
\[x0203bafffe061466/64]    86400   IN     PTR      dmfe1-sun-gw.v6.mydomain.org.
```

The corresponding master configuration file entries are as follows:

```
zone "\[x0000000000000000000000000000000/124].ip6.arpa" in {
      type master;
      file "db..1";
};

zone "\[xfec000a801/64].ip6.arpa" in {
      type master;
      file "db.fec0..a801";
};
```

```
zone "\[xfec000a801/64].ip6.arpa" in {
      type master;
      file "db.fec0..a802";
};
```

At least bitstrings produce a more readable reverse map representation than nibble format.

The last IPv6 DNS extension of RFC-2874 is DNAME. It is only normal for an IPv6 host to have at least two addresses, link-local address and site-local address, and possibly an aggregateable global unicast address as well, yet all these addresses share the same interface identifier. Although it is normal routine to maintain these resource records in one forward zone, the reverse maps will certainly go to three different zones. For example, mysun.v6.mydomain.org may have the addresses found in Table 3:

ADDRESS	DESCRIPTION
fe80:: 203:baff:fe0e:6a6a	Link-local Address
fec0::a801:203:baff:fe0e:6a6a	Site-local Address
2001:11f8:5ef9:a801:203:baff:fe0e:6a6a	Aggregatable Global Unicast Address

Table 3. Possible Addresses for a Single Host

The reverse zone record for the link-local address would be as follows:

```
$ORIGIN \[xfe80/16].ip6.arpa.
\[x000000000000/48]       86400    IN     DNAME          v6-rev.mydomain.org.
```

Similarly, the site-local address would have the following reverse zone record:

```
$ORIGIN \[xfec0/16].ip6.arpa.
\[x00000000a801/48]       86400    IN     DNAME          v6-rev.mydomain.org.
```

The aggregateable global unicast address would follow with a similar entry:

```
$ORIGIN \[x200111f85ef9/48].ip6.arpa.
\[xa801/16]               86400    IN     DNAME          v6-rev.mydomain.org.
```

Finally, the host entry would specify the interface identifier:

```
$ORIGIN v6-rev.mydomain.org.
\[x0203bafffe0e6a6a/64]   86400    IN     PTR            mysun.v6.mydomain.org.
```

DNAME can be used very effectively in reverse maps, since the interface identifier alone contains twice as much information as an IPv4 address. Moreover, renumbering a network may not happen by turning a switch: It requires a lot of careful planning and the transition period can be several months. A reverse map is usually the first thing about the DNS that will be ignored, and, hopefully, DNAME is powerful enough to help the DNS administrators with this area.

Finally, there are some interoperability considerations. Since BIND 8 and especially BIND 8–based resolver libraries are going to be around for a very long time, it is possible to configure BIND 9 to return AAAA records and PTR records in nibble format, even if the zone files use the A6 records and bitstrings. This feature is off by default and can be activated on a per-client basis:

```
options {
      allow-v6-synthesis {
          192.168.1.30;
      };
};
```

RESOLVER

The new BIND 9 Lightweight Resolver consists of two components: the stub library accessed by the applications and the lwresd resolver daemon responsible for processing the forward and reverse lookups (Figure 20). This new setup was mandated by the A6 and DNAME chains, which could have any number of links. The chains are resolved by the daemon, which returns the completed IPv6 address to the resolver library.

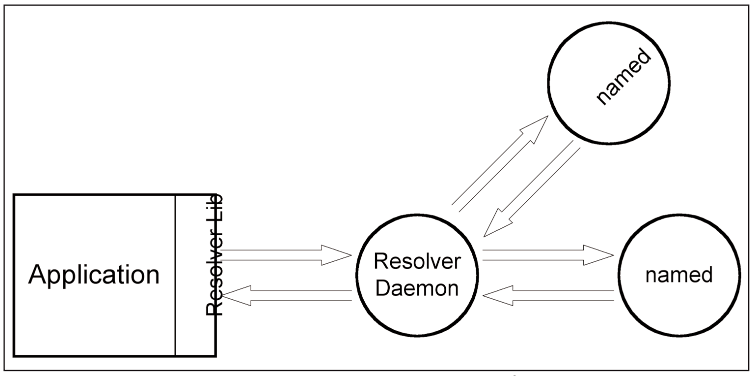


Figure 20. BIND 9 Name Resolution

The lwresd process is essentially a cache-only nameserver for localhost (127.0.0.1, ::1). The communication between the stub library and lwresd is UDP-based at port 921.

IP Security

IP Security, aka IPSec, was originally envisioned as the security architecture for IPv6, but it was retrofitted onto IPv4, mostly for Virtual Private Network tunnels, long before IPv6 had a chance to take off. Consequently, IPSec is almost always associated with VPNs, although it may prove its usefulness in host-to-host communication as well. Good news is that IPSec is a mandatory component of IPv6. The bad news is that we are not there yet.

The general architectural placement of the IPSec layer is between the transport and network layers. The concept of inserting a security protocol between transport and network protocols is actually quite old, since it was one of the ideas proposed in the OSI Security Framework.[26]

The general idea of the IPSec protocol is quite simple, since an IPSec packet consists of two elements: the Authentication Header (AH) and the Encapsulating Security Payload (Figure 21). Since the IPSec protocol takes the transport protocol data unit, authenticates and/or encrypts the data, and passes the completed IPSec packet to the network layer to be transmitted and routed, it can offer very limited protection for the IPv6 protocol. On the other hand, the transport protocol packet (e.g., TCP) enjoys the full protection of the IPSec layer.

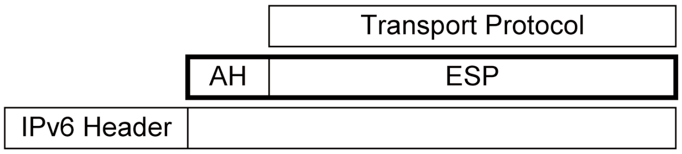


Figure 21. IPSec Packet

AH provides data integrity, data origin authentication, and optional replay protection. It is an authentication service that is used to verify the source of the data and that the data has not been modified while in transit, but it does not protect against eavesdropping. The possible authentication algorithms used by AH are HMAC-MD5 and HMAC-SHA-1.

ESP, on the other hand, offers data confidentiality through encryption. It is possible to combine AH and ESP to have data integrity, data origin authentication, and confidentiality without reapplying AH over an already authenticated packet. The possible encryption protocols are DES, 3DES, Blowfish, and AES. The key lengths are shown below in Table 4.

Since the IPSec architecture is transparent to the application, by definition it cannot be used to authenticate users. The end-user applications such as Telnet, FTP, or X11 must rely on their own methods to verify the authenticity of the users. In other words, if someone is able to establish an IPSec session to the Telnet port and the only authentication mechanism is the standard username and password, the system can be broken into over IPSec. Moreover, if the ESP protects the communication, the network intrusion detection system will never know that something went wrong.

IPSec, and ESP in particular, defeats firewalls too. A firewall cannot see inside the ESP payload and make any kind of filtering decision based on port numbers. In essence, the firewall will

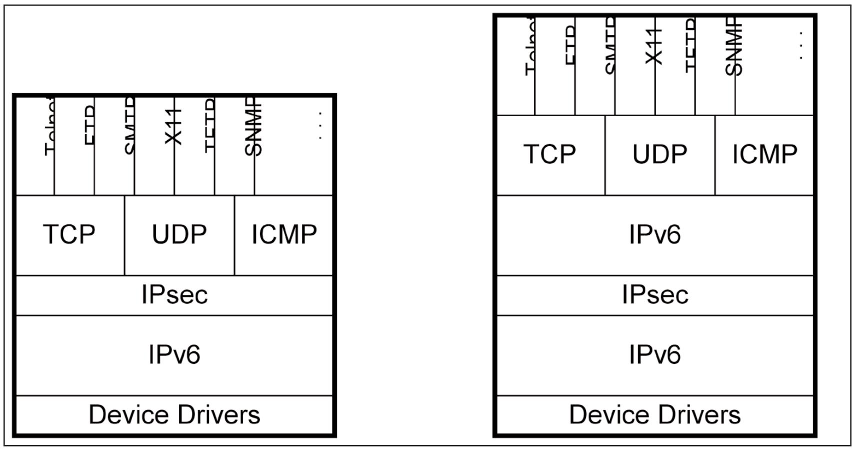


Figure 22. IPSec Transport Mode and Tunnel Mode

only know the end-points (i.e., IPv6 addresses) and has to rely on the host to make the correct decision whether to permit or deny communication for this particular application.

The two modes of IPSec are transport mode and tunnel mode (Figure 22). All VPNs use the tunnel mode, while host-to-host communication should use the transport mode.

The transport mode is meant to protect end-to-end communication between two hosts using IPSec. Since the network analyzers such as snoop and tcpdump attach between the device driver and the network layer, IPSec will protect the data in transit from these as well. Authentication and decapsulation of data takes place after the datagram has been successfully received and processed by the network protocol.

As mentioned earlier, VPNs have been the primary users of IPSec, to the extent that IPSec is often associated with the tunnel mode. Boosted by the VPN market, the tunnel mode has turned into another link layer technology, albeit cheaper than frame relay, ATM, or (fractional) T1. The main security issue with a VPN is to determine whether the other network really is trustworthy.

There have been occasions in which an application needed data integrity and confidentiality. A good example is a query application to a database that contains sensitive financial or other data. Since it may not have been possible to protect the data in transit with SSL/TLS, and the hacks to use SSH as a secure proxy, users were somewhat forced to set up a VPN from the server network to their own group. This kind of setup may raise an eyebrow or two, but such setups are often implemented, since getting the job done is usually more important for a corporate environment than waiting for the one true solution. That one true solution could well be the IPSec transport mode, since it can be configured to protect end-to-end communication to the database application and pass everything else between the hosts in clear. To summarize, the transport mode can put an end to one-application VPNs.

SET IT UP

In the following section we will discuss how to set up KAME IPSec in transport mode to protect traffic between two systems. Since IPSec provides authentication, data integrity, and confidentiality services, we will discuss trivial host authentication for an inbound connection and then move to a more complex setup using both AH and ESP.

By default IPSec is not compiled into the FreeBSD-4.x kernel. Before proceeding with this discussion you may want to check that the following options have been defined in the kernel configuration file:

```
options   IPSEC         #IP security
options   IPSEC_ESP     #IP security (crypto; define w/
                             IPSEC)
options   IPSEC_DEBUG   #debug for IP security
```

You may also want to issue the setkey -D command to find out whether setkey(8) can talk to the IPSec kernel. If you get any kind of error message, you probably don't have IPSec in the kernel. Follow the instructions in the FreeBSD Handbook to compile a new kernel.

Perhaps the trickiest part in IPSec is defining the security associations and the policy controlling the associations. For example, SSH and applications using SSL/TLS are on top of the transport layer, i.e., the transport protocol has already set up a two-way communication channel between the endpoints. The security context is implicit, since it is defined by the underlying TCP connection from Host A to Host B: We know that there is a TCP connection from A to B and we only have to provide a secure channel from A to B to keep the data safe. In contrast, since the IPSec layer is below the transport protocol, all security contexts must be explicitly defined: It is not enough to set up a context from A to B, but one has to define the context from B to A as well.

The various address types of IPv6 add another dimension to the IPSec setup. If a host has a site-local address and an aggregateable global unicast address in addition to the link-local address, all these address types must be specified in the IPSec policy. It would be somewhat silly to have a policy for the link-local address but have no policy whatsoever for the routable addresses. Consequently, an IPSec policy that applies to the routable addresses but omits the link-local one leaves a back door to the host.

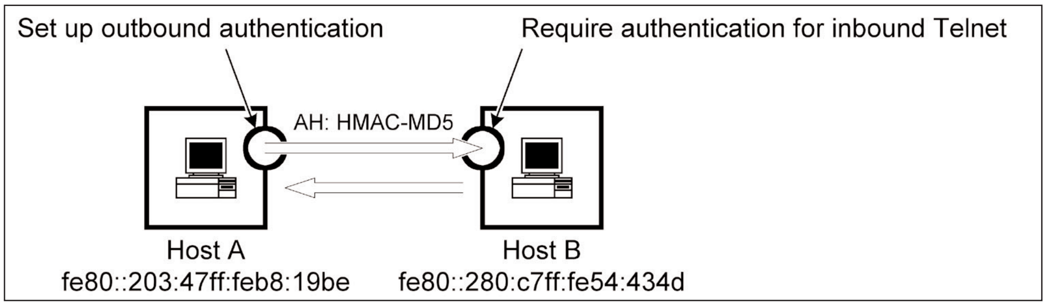


Figure 23. AH required by Host B

HOST AUTHENTICATION

The first example is a simple authentication between two FreeBSD hosts using HMAC-MD5. The link-local address of Host A is fe80::203:47ff:feb8:19be and Host B is fe80::280:c7ff:fe54:434d. For the sake of simplicity we will exclusively use link-local addresses. Any traffic to the Telnet port on Host B must be authenticated at the IPSec layer for further communication to take place. However, no IPSec will apply to the reply packets from B to A. This setup is in Figure 23:

One might find this type of setup suitable to replace, or to use together with, TCP Wrappers. Since TCP Wrappers make the access control decision based on the IP address, one could further strengthen the access control setup by demanding authentication via IPSec before the packet is even passed to TCP Wrappers.

Note that different IPSec algorithms specify different key lengths and that you must have the right key length for the algorithm to accept the key. Table 4 displays the various KAME IPSec authentication and encryption algorithms, and the respective key lengths:

	Key Length			
Algorithm	Bits	Bytes		Description
hmac-md5	128	16		ah: RFC-2403
	128	16		ah-old: RFC-2085
hmac-sha1	160	20		ah: RFC-2404
	160	20		ah-old: 128-bit ICV (no document)
keyed-md5		128	16	ah: rfc1828
		128	16	ah-old: 96-bit ICV (no document)
keyed-sha1		160	20	ah: 96-bit ICV (no document)
		160	20	ah-old: 128-bit ICV (no document)
null		0-2048	0-256	For debugging purposes
hmac-sha2-256		256	32	ah: 96-bit ICV (no document)
		256	32	ah-old: 128-bit ICV (no document)
hmac-sha2-384		384	48	ah: 96-bit ICV (no document
		384	48	ah-old: 128-bit ICV (no document)
hmac-sha2-512		512	64	ah: 96-bit ICV (no document)
		512	64	ah-old: 128-bit ICV (no document)
des-cbc		64	8	esp: RFC-1829
		64	8	esp-old: RFC-2405
3des-cbc		192	24	RFC-2451
simple		0-2048	0-256	RFC-2410
blowfish-cbc		40-448	5-56	RFC-2451
cast128-cbc		40-128	5-16	RFC-2451
des-deriv		64	8	ipsec-ciph-des-derived-01
3des-deriv		192	24	No document
rijndael-cbc		128/192/256	16/24/32	draft-ietf-ipsec-ciph-aes-cbc-00

Table 4. KAME IPSec Authentication Algorithms[27]

1. First we have to define the security associations on A and B in /etc/ipsec.conf. The following example will be used on Host A:

```
# cd /etc
# ex ipsec.conf
"ipsec.conf" [New File]
:a
```

```
add fe80::203:47ff:feb8:19be%fxp0 fe80::280:c7ff:fe54:434d%fxp0 ah 0x30e8 -m transport
                                  -A hmac-md5 "mysecretissecret" ;

spdadd fe80::203:47ff:feb8:19be%fxp0 fe80::280:c7ff:fe54:434d%fxp0 [23] tcp -P out ipsec ah/transport//require ;

.
:wq
"ipsec.conf" [New File] 2 lines, 218 characters written
```

This entry is similar on A and B, with the exception of the interface suffix (i.e., fxp0 on Host A and xe0 on Host B). Note that each statement must fit on one line. Line feeds and line escapes (\) are not permitted. The hanging indent in the example above is only for typographical reasons.

The syntax for add and spdadd can be found from the setkey(8) manual page. Note that the Security Parameter Index (0x3048) for add should be a random number, which has to be greater than or equal to 256. This number will be passed between A and B to determine the correct security association between the systems. And since the AH algorithm is HMAC-MD5, the shared key must be 16 bytes long.

The spdadd entry will define the security policy for the security association. The security association is between the hosts A and B, and the association will become active when A opens a TCP connection to the port 23 (Telnet) on the Host B. All other communication (e.g., FTP, TFTP, or ping) will take place outside IPSec.

2. Load the key and the policy into the operating system kernel with the setkey(8):

```
# setkey -f /etc/ipsec.conf
```

3. Set up Host B similarly to require AH authentication for inbound traffic:

```
# cd /etc
# ex ipsec.conf
"ipsec.conf" [New File]
:a
add fe80::203:47ff:feb8:19be%fxp0 fe80::280:c7ff:fe54:434d%fxp0 ah 0x30e8 -m transport
                              -A hmac-md5 "mysecretissecret" ;

spdadd fe80::203:47ff:feb8:19be%fxp0 fe80::280:c7ff:fe54:434d%fxp0 [23] tcp -P out ipsec ah/transport//require ;

.
:wq
"ipsec.conf" [New File] 2 lines, 218 characters written
```

Of course, instead of specifying Host A for spdadd one could use the IPv6 wildcard address (::) to require AH for all connection attempts to the Telnet port.

4. Load the configuration into the kernel on Host B:

```
# setkey -f /etc/ipsec.conf
```

5. Once the IPSec configuration is active, you can view the security association database with setkey -D:

```
# setkey -D
fe80::203:47ff:feb8:19be%xe0 fe80::280:c7ff:fe54:434d%xe0
    ah mode=any spi=12520(0x000030e8) reqid=0(0x00000000)
    A: hmac-md5  6d797365 63726574 6d797365 63726574
    seq=0x0000000e replay=0 flags=0x00000040 state=mature
    created: Sep 22 14:42:32 2002     current: Sep 22 14:49:27 2002
    diff: 415(s           hard: 0(s)           soft: 0(s)
    last: Sep 22 14:42:44 2002       hard: 0(s)   soft: 0(s)
    current: 1184(bytes)    hard: 0(bytes)     soft: 0(bytes)
    allocated: 1          hard: 0       soft: 0
    sadb_seq=2 pid=640 refcnt=1
```

On the other hand, the -DP command will show the effective IPSec policies. There should be only one entry since we didn't have anything else in /etc/ipsec.conf:

```
# setkey -DP
fe80::203:47ff:feb8:19be%xe0 [any] fe80::280:c7ff:fe54:434d%xe0[23] tcp
    in ipsec
    ah/transport//require
    spid=45 seq=1 pid=641
    refcnt=1
```

6. You can test the setup by opening a Telnet session from Host A to B:

```
# telnet -K fe80::280:c7ff:fe54:434d%fxp0
Trying fe80::280:c7ff:fe54:434d%fxp0...
Connected to fe80::280:c7ff:fe54:434d%fxp0
Escape character is '^]'.

FreeBSD/i386 (hostb) (ttyp2)

login:
```

We can monitor the progress of the connection with tcpdump. The first two packets exchanged at the beginning of the session are as follows:

```
fe80::203:47ff:feb8:19be > fe80::280:c7ff:fe54:434d: AH(spi=0x000030e8,seq=0xf): 1052 > 23: S
        3462415989:3462415989(0) win 16384 <mss 1440,nop,wscale 0,nop,nop,timestamp 527422 0>
fe80::280:c7ff:fe54:434d.23 > fe80::203:47ff:feb8:19be.1052: S 1868912997:1868912997(0) ack 3462415990
        win 57344 <mss 1440,nop,wscale 0,nop,nop,timestamp 752453 527422>
```

Obviously, traffic from Host A to Host B contains the AH header with the Security Parameter Index of 0x30e8, and the acknowledgments from B to A are normal TCP packets. If you unloaded the policy on B or changed the key on either host, it would be impossible for Host B to connect:

```
# telnet -K fe80::280:c7ff:fe54:434d%fxp0
Trying fe80::280:c7ff:fe54:434d%fxp0...
```

Although having host-based authentication before the connection is really opened sounds like a good idea, there are at least two security issues with this setup. First, the Security Parameter Index is static. Our packet capture with tcpdump(8) shows clearly how the SPI is exchanged between the hosts in the clear. The ideal SPI is a large random number that changes for each security association. Similarly, the shared secret that is used over the connection is static, and it is stored into /etc/ipsec.conf and the security association database in cleartext.

AUTHENTICATE AND ENCRYPT DATA BETWEEN TWO HOSTS

In the second example we will secure all TCP communication between Host B running FreeBSD-4.x and Host C running Solaris 9. Before we can proceed, it is a good idea to flush the setup on Host B to avoid surprises:

```
# setkey -FP
# setkey -F
```

In this setup we will encrypt the payload, i.e., TCP packet, using 3DES-CBC in the ESP, and we will enforce data integrity with HMAC-MD5 in the AH. To make this setup more interesting, there will be one pair of shared keys, one for AH and another one for the ESP, for the direction Host C to Host B, and another key pair for traffic from Host B to Host C. This setup is shown in Figure 24 on the next page.

We will not use key management in this example, since in.iked(1m), the Solaris 9 IKE daemon, only works with IPv4.[28] Moreover, the KAME IKE daemon, racoon(8),[29] must be built separately from FreeBSD ports. To summarize, it may take a while before IKE is universally available for the various IPv6 IPSec implementations.

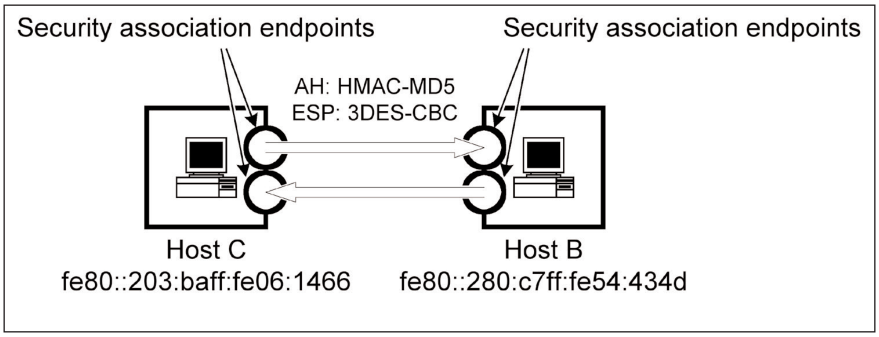


Figure 24. Security Association with AH and ESP

1. First we have to define the security associations and policies on FreeBSD. You may remove the old /etc/ipsec.conf and start working on a new one:

lines below broken for display

```
# rm ipsec.conf
# ex ipsec.conf
"ipsec.conf" [New File]
:a
add fe80::203:baff:fe06:1466%xe0 fe80::280:c7ff:fe54:434d%xe0 esp 0xe63f -E
               3des-cbc "JnRs6Riou3wwlPHv5zdPED5k" -A hmac-md5 "hostahostbsecret" ;
add fe80::280:c7ff:fe54:434d%xe0 fe80::203:baff:fe06:1466%xe0 esp 0xf52a -E
               3des-cbc "G95lvYfQePVkul9Byvibi7fd" -A hmac-md5 "hostbhostasecret" ;

spdadd fe80::203:baff:fe06:1466%xe0 fe80::280:c7ff:fe54:434d%xe0 tcp -P in ipsec esp/transport//require ;
spdadd fe80::280:c7ff:fe54:434d%xe0 fe80::203:baff:fe06:1466%xe0 tcp -P out ipsec esp/transport//require ;
.
:wq
```

Since the key length for 3DES-CBC is 192 bits, our shared keys must be 24 characters long. One should always use random strings as the keys, but for demonstration purposes we selected simple keys for authentication. In this case the encryption keys are random strings since Solaris ipseckeys(1m) would otherwise complain about weak keys.

2. Load the policy into the kernel:

```
# setkey -f /etc/ipsec.conf
```

3. Print out the entries in the security association database:

```
# setkey -D
fe80::280:c7ff:fe54:434d%xe0 fe80::203:baff:fe06:1466%xe0
    esp mode=any spi=62762(0x0000f52a) reqid=0(0x00000000)
    E: 3des-cbc  4739356c 76596651 6550566b 756c3942 79766962 69376664
    A: hmac-md5  686f7374 62686f73 74617365 63726574
    seq=0x0000003b replay=0 flags=0x00000040 state=mature
    created: Sep 24 15:05:37 2002current: Sep 24 15:17:35 2002
    diff: 718(s)     hard: 0(s)       soft: 0(s)
    last: Sep 24 15:14:15 2002       hard: 0(s)                soft: 0(s)
    current: 6484(bytes)        hard: 0(bytes      soft: 0(bytes)
    allocated: 59       hard: 0       soft: 0
    sadb_seq=1 pid=311 refcnt=2
fe80::203:baff:fe06:1466%xe0 fe80::280:c7ff:fe54:434d%xe0
    esp mode=any spi=58943(0x0000e63f) reqid=0(0x00000000)
    E: 3des-cbc  4a6e5273 3652696f 75337777 6c504876 357a6450 4544356b
    A: hmac-md5  686f7374 61686f73 74627365 63726574
    seq=0x00000041 replay=0 flags=0x00000040 state=mature
    created: Sep 24 15:05:37 2002     current: Sep 24 15:17:35 2002
    diff: 718(s)      hard: 0(s)        soft: 0(s)
    last: Sep 24 15:14:15 2002       hard: 0(s)                soft: 0(s)
    current: 4801(bytes)         hard: 0(bytes)     soft: 0(bytes)
    allocated: 65       hard:      0 soft: 0
    sadb_seq=0 pid=311 refcnt=1
```

This command will dump the contents of the security administration database on-screen, including the AH and ESP keys in the hexadecimal format. Since we cannot enter ASCII keys into the Solaris IPSec key database (/etc/inet/secret/ipseckeys), we must copy the entries to the Solaris key file. This raises the age-old question of how to arrange secure key distribution to avoid sending the AH and the ESP keys over an insecure connection. SSH is always one solution, but it only provides a channel: The key distribution process is still largely manual work or requires custom scripts to extract shared secrets from the central host and distribute those to the remote hosts.

4. On Solaris, edit /etc/inet/secret/ipseckeys:

```
  # cd /etc/inet/secret
  # ex ipseckeys
  "ipseckeys" [...]
  :a
  add esp spi 0xf52a src6 fe80::280:c7ff:fe54:434d dst6 fe80::203:baff:fe06:1466 \
      encralg 3des-cbc encrkey 4739356c765966516550566b756c39427976696269376664 \
      authalg hmac-md5 authkey 686f737462686f737461736563726574
  add esp spi 0xe63f src6 fe80::203:baff:fe06:1466 dst6 fe80::280:c7ff:fe54:434d \
      encralg 3des-cbc encrkey 4a6e52733652696f753377776c504876357a64504544356b \
      authalg hmac-md5 authkey 686f737461686f737462736563726574
.
:wq
```

5. Now we have to load the keys into the Solaris IPSec kernel. This is accomplished with the ipseckey(1m) command:

```
  # ipseckey -f ipseckeys
```

We can use ipseckey also to verify that the entries were loaded correctly:

```
  # ipseckey -n dump
  Base message (version 2) type DUMP, SA type ESP.
  Message length 200 bytes, seq=1, pid=8407.
  SA: SADB_ASSOC spi=0xe63f, replay=0, state=MATURE
  SA: Authentication algorithm = HMAC-MD5
  SA: Encryption algorithm = 3DES-CBC
  SA: flags=0x80000000 < X_USED >
  SRC: Source address (proto=0)
  SRC: AF_INET6: port 0, fe80::203:baff:fe06:1466.
  DST: Destination address (proto=0)
  DST: AF_INET6: port 0, fe80::280:c7ff:fe54:434d.
  AKY: Authentication key.
  AKY: 686f737461686f737462736563726574/128
  EKY: Encryption key.
  EKY: 4a6e52733752686e753276766d514976347a64514545346b/192
  LT: Lifetime information
  CLT: 3072 bytes protected, 0 allocations used.
  CLT: SA added at time Tue Sep 24 14:09:01 2002
  CLT: SA first used at time Tue Sep 24 14:14:10 2002
  CLT: Time now is Tue Sep 24 14:19:24 2002
```

```
Base message (version 2) type DUMP, SA type ESP.
Message length 200 bytes, seq=1, pid=8407.
SA: SADB_ASSOC spi=0xf52a, replay=0, state=MATURE
SA: Authentication algorithm = HMAC-MD5
SA: Encryption algorithm = 3DES-CBC
SA: flags=0x0 < >
SRC: Source address (proto=0)
SRC: AF_INET6: port 0, fe80::280:c7ff:fe54:434d.
DST: Destination address (proto=0)
DST: AF_INET6: port 0, fe80::203:baff:fe06:1466.
AKY: Authentication key.
AKY: 686f737462686f737461736563726574/128
EKY: Encryption key.
EKY: 4638346d765867516451576b756d38437976686268376764/192
LT: Lifetime information
CLT: 2944 bytes protected, 0 allocations used.
CLT: SA added at time Tue Sep 24 14:09:01 2002
CLT: Time now is Tue Sep 24 14:19:24 2002

Dump succeeded for SA type 0.
```

Now we only need the policy to control the use of IPSec.

6. Edit /etc/inet/ipsecinit.conf:

```
# cd /etc/inet
# ex ipsecinit.conf
:a
{
    laddr fe80::203:baff:fe06:1466
    raddr fe80::280:c7ff:fe54:434d
    ulp tcp
} ipsec {
    encr_algs 3des-cbc
    encr_auth_algs hmac-md5
    sa shared
}
.
:wq
```

In this file we will specify that one has to use 3DES-CBC encryption with HMAC-MD5 authentication for all TCP communication between Hosts B and C. Since the direction has not been specified, the policy will apply to both directions. The syntax of ipsecconf(1m) allows more complex policy setups, but these have already been documented in the manual page.[30]

7. Load the IPSec policy into the kernel:

```
# ipsecconf -a ipsecinit.conf
```

Now we are all set. If there were no typos in the shared keys or the Security Parameter Indices, all TCP traffic between B and C is protected with IPSec. As usual, the easiest way to find out is to launch Telnet and monitor the connection with tcpdump (FreeBSD) or snoop (Solaris). If everything went well, the connection attempt from B to C will look like this:

```
# telnet fe80::280:c7ff:fe54:434d
Trying fe80::280:c7ff:fe54:434d...
Connected to fe80::280:c7ff:fe54:434d.
Escape character is '^]'.

FreeBSD/i386 (hostb) (ttyp2)

login:
```

The output from tcpdump should show something like this for the first two packets:

```
fe80::203:baff:fe06:1466 > fe80::280:c7ff:fe54:434d: ESP(spi=0x0000e63f,seq=0x42)
0x0000     6000    0000    003c    323c    fe80    0000    0000    0000    `....<2<........
0x0010     0203    baff    fe06    1466    fe80    0000    0000    0000    .......f........
0x0020     0280    c7ff    fe54    434d    0000    e63f    0000    0042    .....TCM...?...B
0x0030     4e9c    e2dd    3a98    76d4    5eca    1717    3b1c    9538    N...:.v.^...;..8
0x0040     2b0f    5f74    0056    71f7    34f5    1923    a694    68f6    +._t.Vq.4..#..h.
0x0050     d7f0    1c20    3543    ee5e    8bdc    8fe6    09cb    7ab1     ....5C.^......z.
0x0060     9b7a    546f    5cbc    b702                                    .zTo\...
fe80::280:c7ff:fe54:434d > fe80::203:baff:fe06:1466: ESP(spi=0x0000f52a,seq=0x3c) [flowlabel 0x3baff]
0x0000     6003    baff    003c    3240    fe80    0000    0000    0000    `....<2@........
0x0010     0280    c7ff    fe54    434d    fe80    0000    0000    0000    .....TCM........
0x0020     0203    baff    fe06    1466    0000    f52a    0000    003c    .......f...*...<
0x0030     1db8    733e    f397    3c1f    7fef    083d    39d8    eda6    ..s>..<....=9...
0x0040     1b3d    d8a2    1b3e    c8d5    b149    e432    4e89    c31d     .=...>...I.2N...
0x0050     bdb4    c7c5    101b    bafb    19a5    cf0f    d358    e354    .............X.T
0x0060     1753    0af8                                                    .S..
```

When we used tcpdump earlier to monitor AH packets, it was able to disassemble the AH header, peek into the TCP packet, and show us the sequence numbers and other TCP parameters. However, since ESP completely hides the upper-layer payload, we would not know anything about the upper-layer protocol if we hadn't configured the IPSec policy by ourselves. In other words, we can only see that fe80::203:baff:fe06:1466 sent a packet to fe80::280:c7ff:fe54:434d and the receiving host seemed to acknowledge it. It should be no surprise that a packet like this will make a firewall and the (network) intrusion detection very angry.

IPSec is a very powerful network security mechanism to protect applications from network eavesdropping or interception. This discussion has been in the context of IPv6, but the main user community for IPSec seems to be in IPv4, at least for the time being. Network and enterprise management is likely to be the primary beneficiary of IPSec, since management applications have had little or no security capabilities. The list of dangerously insecure million-dollar enterprise systems seems endless and any responsible enterprise management project should absolutely take a close look at the transport mode and assess whether it could be used to improve the security of the management system.

As with AH earlier, the primary security issues with the setup described above are the static Security Parameter Indices and static shared keys. Both SPI and the key should be random values that can be disposed of at the end of the session. History shows repeatedly it is a bad practice to store cleartext secrets in configuration files, even if the files are only readable by the super-user. One can decrease the risk by issuing unique keys for each security association (e.g., four keys for a bi-directional security association), but clearly this creates a key distribution problem.

Conclusion

IPv6 has come a long way from the drafts to the current implementations, but there is still a lot of work to be done. Although IPv6 has been around in one form or another for a long time, it may take a while before IPv6 will be mainstream. This is underscored by the facts that one must run Solaris 9 to get IPSec to work with IPv6, or that only Windows XP users can run Microsoft's IPv6 stack. One must also have BIND 9 to be able to use the new A6 and DNAME resource records, which all means that it will take years before bleeding-edge technology becomes commodity.

Yet IPv6 is a great protocol. When I set up my first IPv6 nodes I had to ask myself several times why I had waited so long. The addressing plans with link-local, site-local, and aggregateable global unicast addresses and stateless address auto-configuration made a lot more sense than requesting that an IP address become eligible in order to get connected to the network.

On the other hand, a new network protocol isn't any good if the only applications for it are ports of a 20-year-old virtual terminal or a file transfer program. Luckily, the situation is not really that bad for IPv6. Obviously, Telnet and FTP were the first IPv6 applications since, they have been a part of TCP/IP for ages. There are IPv6 ports available for open source applications at *ftp.kame.net* and the FreeBSD Ports Collection, for example, shows an ever-growing list of IPv6-enabled applications.

IPv6 is a successor of IPv4 in the sense that it is not very difficult to get IPv4 applications to speak IPv6. There are socket scrubbers available to identify IPv4-specific code and suggest changes to make them IPv6-ready. Consequently, there are few IPv6-only applications, and the convention seems to be to simply make the applications listen to IPv4 and IPv6 sockets simultaneously. This approach does indeed give the users more flexibility in deciding their migration path.

Even if IPv6 networks are hard to find, the developers could start using IPv6 immediately. Since the startup cost to enable IPv6 is minimal – the code is in the operating system already and enabling it requires two steps – it is actually the new interprocess communication mechanism. You don't even need a router to run IPv6 on the local LAN, since the link-local addresses are available to everyone anyway. IPv6 is not a unique protocol in the sense that many IPv6 features are available for IPv4 as well, starting with IPSec. Yet IPv6 does not have to carry the history of IPv4; the many good ideas and concepts arising from IPv4 have been imported to IPv6, and now it is time to move on.

References

[1] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC-2460, December 1998. Available from *http://www.ietf.org/rfc/rfc2460.txt*.

[2] J. Reynolds and J. Postel, "Assigned Numbers," RFC-1700, October 1994. Available from *http://www.ietf.org/rfc/rfc1700.txt* and *http://www.iana.org/*.

[3] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC-2373, July 1998. Available from *http://www.ietf.org/rfc/rfc2373.txt*.

[4] IPv6 Address Oracle, Indiana University, Advanced Network Management Laboratory. Available from *http://steinbeck.ucs.indiana.edu:47401/*.

[5] C. Huitema, *IPv6: The New Internet Protocol*, 2d ed., Prentice-Hall, 1998.

[6] IEEE, "Guidelines for 64-bit Global Identifier (EUI-64™) Registration Authority," May 2001. Available from *http://standards.ieee.org/regauth/oui/tutorials/EUI64.html*.

[7] D. Plummer, "An Ethernet Address Resolution Protocol," RFC-826, November 1982. Available from *http://www.ietf.org/rfc/rfc0826.txt*.

[8] IANA, "Special-Use IPv4 Addresses," IETF Network Working Group, August 2002. Available from *http://www.ietf.org/internet-drafts/draft-iana-special-ipv4-05.txt*.

[9] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets," RFC-1918, February 1996. Available from *http://www.ietf.org/rfc/rfc1918.txt*.

[10] J. Itoh, "Overview of KAME project," Work-in-progress report, USENIX 1998 Annual Conference, New Orleans, 1998. Available from *http://www.kame.net/project-overview.html*.

[11] R. Hinden, M. O'Dell, and S. Deering, "An IPv6 Aggregatable Global Unicast Address Format," RFC-2374, July 1998. Available from *http://www.ietf.org/rfc/rfc2373.txt*.

[12] R. Hinden, S. Deering, R. Fink, and T. Hain, "Initial IPv6 Sub-TLA ID Assignments," RFC-2928, September 2000. Available from *http://www.ietf.org/rfc/rfc2928.txt*.

[13] Global IPv6 Allocations Made by the Regional Internet Registries, RIPE. Available from *http://www.ripe.net/ipv6/ipv6allocs.html*.

[14] B. Carpenter, K. Moore, and B. Fink, "Connecting IPv6 Routing Domains over the IPv4 Internet," *Cisco Internet Protocol Journal*, vol. 3, no. 1, March 2000. Available from *http://www.ieng.com/warp/public/759/ipj_3-1/ipj_3-1_routing.html*.

[15] B. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds," RFC-3056, February 2001. Available from *http://www.ietf.org/rfc/rfc3056.txt*.

[16] S. Deering, "Host Extensions for IP Multicasting," RFC-1112, August 1989. Available from *http://www.ietf.org/rfc/rfc1112.txt*.

[17] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC-2462, December 1998. Available from *http://www.ietf.org/rfc/rfc2462.txt*.

[18] Sun Microsystems, "Administering IPv6," Solaris System Administration Guide: IP Services, part no. 806-4075–10, pages 303-322, May 2002. Available from *http://docs.sun.com/*.

[19] R. Zilbauer, "Configuring IPv6 on Solaris 8," Native6Group, 2002. Available from *http://www.native6group.com/docs/Configuring%20IPv6%20on%20Solaris8.pdf*.

[20] Sun Microsystems, "ipnodes – Local Database Associating Names of Nodes with IP Addresses," Solaris 9 Reference Manual Collection, Section 4: File Formats, October 1999. Available from *http://docs.sun/com/*.

[21] B. Bucklin and Y. Sekiya, "IPv6 DNS Setup Information," January 31, 2000. Available from *http://www.isi.edu/~bmanning/v6DNS.html*.

[22] K. Harrenstien, M. Stahl, and E. Feinler, "DoD Internet Host Table Specification," RFC-952, October 1985. Available from *http://www.ietf.org/rfc/rfc0952.txt*.

[23] S. Thomson and C. Huitema, "DNS Extensions to Support IP Version 6," RFC-1886, December 1995. Available from *http://www.ietf.org/rfc/rfc1886.txt*.

[24] M. Crawford and C. Huitema, "DNS Extensions to Support IPv6 Address Aggregation and Renumbering," RFC-2874, July 2000. Available from *http://www.ietf.org/rfc/rfc2874.txt*.

[25] Internet Software Consortium, "IPv6 Support in BIND 9," BIND 9 Administrator Reference Manual, pages 36–39, 2001. Available from *http://www.nominum.com/resources/documentation/Bv9ARM.pdf*.

[26] ISO, "OSI Security Model, Part 1: Security Framework," ISO 10181-1.

[27] KAME Project, "setkey – Manually Manipulate the Ipsec SA/SP Database," FreeBSD System Manager's Manual, Chapter 8, FreeBSD, 2000. Available from *http://www.freebsd.org/cgi/man.cgi?query=setkey&apropos=0&sektion=8&manpath=FreeBSD+4.6-RELEASE&format=html*.

[28] Sun Microsystems, "in.iked – Daemon for the Internet Key Exchange (IKE)," Solaris 9 Reference Manual Collection, Section 1m: System Administration Commands, Sun Microsystems, Inc., February 2002. Available from *http://docs.sun/com/*.

[29] KAME Project, "racoon – IKE (ISAKMP/Oakley) key management daemon," available in the KAME distribution from *http://www.kame.net/*.

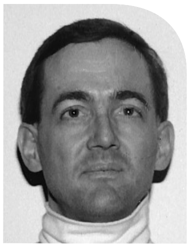
[30] Sun Microsystems, "ipsecconf – configure system wide IPSec policy," Solaris 9 Reference Manual Collection, Section 1m: System Administration Commands, February 2002. Available from *http://docs.sun/com/*.

examining the C# "hello, world" program

by Glen McCluskey

Glen McCluskey is a consultant with 20 years of experience and has focused on programming languages since 1988. He specializes in Java and C++ performance, testing, and technical documentation areas.

glenm@glenmccl.com



In our introductory C# column, we discussed some of the background and context for the C# language, in particular how C# fits into the .NET Framework. In this column we'll start describing the C# language itself.

The Hello Program

Let's look first at the C# version of the "Hello, world" program. This is a trivial application but one we can use to tie down many of the C# basics. Here's the code:

```
using System;

class Hello {
    static void Main() {
        Console.WriteLine("Hello, world!");
    }
}
```

We're going to assume the use of the Microsoft SDK in our discussions. Given this, we compile and execute the program by saying:

```
$ csc Hello.cs
$ Hello
```

The compilation produces an EXE file. This file is very small (3072 bytes) and is not self-contained. It depends on the presence on the local system of the .NET JIT (just-in-time compiler), as discussed in the introductory column. The csc compiler generates opcodes in an intermediate language, and the opcodes are stored in the EXE. When the EXE is executed at some later time, the JIT is used to translate the intermediate language into machine language.

We put our Hello program in a source file Hello.cs. But there's no file-naming requirement implied by doing so; we can instead say:

```
$ cp Hello.cs xyz.cs
$ csc xyz.cs
$ xyz
```

and it will still work.

Another point about this example is that there is a distinguished method named Main() in a C# program, used as the entry point for program execution. Main is a static method in the Hello class, "static method" meaning that the Main method does not operate on instances of the Hello class but is part of the class for packaging purposes. In this particular example, we don't actually create any Hello class objects. For languages like C++ and C#, using a class as a packaging vehicle for static methods and data is a common program-structuring technique.

Classes are a basic unit of composition and design in C#. C# programming consists of the development of new types, realized via classes and the related struct and interface mechanisms, along with use of standard types such as System.String.

The Main method is called to begin execution of the Hello program, and there's a single statement to execute. Console is a class found in the System namespace, and the "using System" statement says that the types found in this namespace are made available to the Hello program. If we got rid of the using statement, we'd need to say:

```
System.Console.WriteLine("Hello, world!");
```

Using statements is very convenient but can sometimes pollute the application with extraneous and conflicting names.

Namespaces are another basic design mechanism for C# programs; they serve to collect and segregate names in a large application. In our example, we refer to the System namespace. There's also a single global namespace to which the Hello type is added, given that we don't specify a namespace for the Hello class. We could instead say:

```
namespace ABC {
    class Hello {
        ...
    }
}
```

to put the Hello class into the ABC namespace.

Actual output from the Hello program is done by the WriteLine method. This is a static method that is part of the Console class found in the System namespace. It writes output to standard output and is equivalent to:

```
Console.Out.WriteLine("Hello, world!");
```

in which the stream (In, Out, Error) is specified. I/O occurs using standard streams, and I can redirect the output in the usual way:

```
$ Hello > out
```

For the statement:

```
System.Console.Out.WriteLine("Hello, world!");
```

System is a namespace, Console a class in that namespace, Out a static stream object of class TextWriter in the Console class, and WriteLine a method in the TextWriter class.

Command-Line Arguments and Exit Codes

Let's go on and look at another program, this one a variation of the Hello program. It illustrates some additional C# features.

Suppose that you'd like to write the Hello program, but instead of the output going to standard output, you want to specify the output file on the command line, and have output written to that file. How can you do this? Here's an example:

```
using System;
using System.IO;

class FileIO {
      static void Main(string[] args) {
          if (args.Length != 1) {
              Console.Error.WriteLine("missing filename");
              Environment.Exit(1);
          }

          try {
              StreamWriter sw = new StreamWriter(args[0]);
              sw.WriteLine("Hello, world!");
              sw.Close();
          }
          catch {
              Console.Error.WriteLine("couldn't open file");
              Environment.Exit(1);
          }
      }
}
```

Command-line arguments are represented by an array of strings. If there's no command-line argument specified, the program writes an error message to standard error and exits with a non-zero status.

Otherwise, the StreamWriter class is used to perform output to a file. Note that StreamWriter operations are wrapped in a try/catch block. This is done because C# uses exceptions to signal error conditions: for example, failure to open a text file for writing. We use the try/catch block to catch any exception that is thrown. If there is no try/catch block, and an invalid file is specified, the program will terminate with an unhandled-exception diagnostic.

Character Encodings

C# uses the Unicode 16-bit character set. However, in our examples so far, we've implicitly assumed that ASCII is being used: for example, when redirecting output to a file. How does this work? C# I/O uses encodings to map Unicode into other character sets. For example, when you run this little program:

```
using System;
using System.IO;

class Encode {
      static void Main() {
          StreamWriter sw = new StreamWriter("out");
          Console.WriteLine("file encoding is: "
                + sw.Encoding);
          sw.Close();
      }
}
```

the result is:

```
file encoding is: System.Text.UTF8Encoding
```

Roughly speaking, the UTF-8 encoding maps Unicode characters with 7-bit ASCII values into the corresponding ASCII characters, and other Unicode characters into two or three bytes. So C# can use Unicode and still be compatible with ASCII. A similar mechanism is used in Java I/O.

Applications with Multiple Source Files

Suppose that you'd like to write the Hello program, but as part of a more elaborate system whereby messages are logged to a file along with a timestamp. How might this be done?

In such a case it is worth defining your own C# class. Instances of the class represent an open file, to which messages are being logged. Here's what the code looks like:

```
// MainFile.cs

class MainFile {
      static void Main(string[] args) {
          FileLogger flog = new FileLogger(args[0]);
          flog.write("Hello, world!");
          flog.close();
      }
}

// FileLogger.cs

using System;
using System.IO;

class FileLogger {
      private StreamWriter sw;

      public FileLogger(string fn) {
          sw = new StreamWriter(fn);
      }

      public void write(string msg) {
          sw.WriteLine(DateTime.Now + ":  " + msg);
      }
```

```
        public void close() {
            sw.Close();
        }
    }
```

You compile this program by saying:

```
  $ csc MainFile.cs FileLogger.cs
```

Compilation units can depend on each other. For example, the MainFile class uses the FileLogger class, defined in a separate file.

When you run the program, by saying:

```
  $ MainFile outlog
```

the result written to the log file is something like this:

```
  11/29/2002 10:33:35 AM:  Hello, world!
```

This particular application does not handle exceptions via try/catch. If you specify an invalid log file name, for example:

```
  $ MainFile .
```

the program will abort, and display a stack traceback for the unhandled exception.

Browsing System Types

There's one final area we'd like to look at in our discussion of the C# Hello program. Earlier we mentioned the System.Console class, a standard class used for console I/O. How can you know what the standard classes are? Obviously, you can consult reference books, browse online documentation, and look at Web sites.

But there's another way to find out what standard types are available, using the reflection features of C#. You can write a C# program that essentially asks itself what types it knows about and displays the names of those types.

Here's some code that does this:

```
  using System;
  using System.Reflection;

  class DumpTypes {

      // Display a help message and exit.

      private static void dohelp() {
          Console.WriteLine("Usage: [-h|-help] " +
              "[-dumpmem|-m] [-t|-type targ_type] patt1 patt2 ...");
          System.Environment.Exit(0);
      }

      // Filter a string to determine if it contains
      // patterns specified on the command line.

      private static bool filter(string str, string[] args, int argi) {
          str = str.ToLower();

          for (int i = argi; i < args.Length; i++) {
              if (str.IndexOf(args[i]) == -1)
                  return false;
          }

          return true;
      }

      public static void Main(string[] args) {
          bool mem_flag = false;
          string targ_type = null;

          // Parse command-line arguments.

      int argi = 0;
          while (argi < args.Length && args[argi][0] == '-') {
              if (args[argi] == "-dumpmem" || args[argi] == "-m") {
                  mem_flag = true;
              }
              else if (args[argi] == "-t" || args[argi] == "-type") {
                  if (argi + 1 < args.Length) {
                      targ_type = args[argi + 1].ToLower();
                      argi++;
                  }
              }
              else if (args[argi] == "-h" || args[argi] == "-help") {
                  dohelp();
              }

              argi++;
          }
          for (int i = argi; i < args.Length; i++)
              args[i] = args[i].ToLower();

          // Get a list of all types found in the standard
          // library.

          Assembly asm = Assembly.Load("mscorlib.dll");
          Type[] typelist = asm.GetTypes();

          // Iterate across each type.

          foreach (Type atype in typelist) {

              // Check whether type matches specified
              // target type.

              string typestr = atype.ToString();
              if (targ_type != null && typestr.ToLower() !=
                          targ_type)
                  continue;

              // If asked to display all members, iterate
              // across them.

              if (mem_flag) {
                  MemberInfo[] memlist = atype.GetMembers();
```

```
                foreach (MemberInfo amember in memlist) {
                    string memstr = amember.ToString();
                    string s = typestr + " " + memstr;
                    if (filter(s, args, argi))
                        Console.WriteLine("{0}   {1}",
                            typestr, memstr);
                }
            }

            // Just display the type itself without
            // members.

            else {
            if (filter(typestr, args, argi))
                Console.WriteLine(typestr);
            }
        }
    }
}
```

The source code for this program and others in this column is available at *ftp://ftp.glenmccl.com/pub/usenix/cs2.zip*.

The heart of the DumpTypes program is these two lines:

```
Assembly asm = Assembly.Load("mscorlib.dll");
Type[] typelist = asm.GetTypes();
```

An assembly is a collection of files something like a shared library or archive. These lines of code load the standard C# assembly and extract a list of types from it, using the GetTypes method. Given a type such as a class and it's possible to call an analogous method (GetMembers) to find a list of the members (methods, fields, properties) defined within the type.

If you run this program with no command-line arguments, it displays a list of standard types:

```
System.Object
System.ICloneable
System.Collections.IEnumerable
System.Collections.ICollection
System.Collections.IList
...
```

If you specify a given type:

```
$ DumpTypes -t System.String
```

it displays just that type. If you specify that members are to be displayed as well, like this:

```
$ DumpTypes -t System.String -m
```

the result is a list of members for the System.String class:

```
System.String   System.String Empty
System.String   System.String ToString(System.IFormatProvider)
System.String   System.TypeCode GetTypeCode()
System.String   System.Object Clone()
System.String   Int32 CompareTo(System.Object)
...
```

For example, the first line of output refers to the Empty static field of System.String. Empty is of type System.String and refers to an empty string ("").

If you specify a list of matching patterns, every line of output will be filtered against all those patterns. For example, saying:

```
$ DumpTypes -m
```

provides a voluminous list of all types and members, a total of more than 20,000 lines of output. But if you say:

```
$ DumpTypes -m cos
```

the output is:

```
System.Math    Double Acos(Double)
System.Math    Double Cos(Double)
System.Math    Double Cosh(Double)
```

which describes three trigonometric methods in the System.Math class.

We've looked at some of the basics around writing C# programs. C# is typically pitched as a language for developing GUI and network applications, but you can also write stand-alone utility programs just as you would in C programming.

the tclsh spot

by Clif Flynt

Clif Flynt is president of Noumena Corp., which offers training and consulting services for Tcl/Tk and Internet applications. He is the author of *Tcl/Tk for Real Programmers* and the *TclTutor* instruction package. He has been programming computers since 1970 and a Tcl advocate since 1994.

clif@cflynt.com

The previous two Tclsh Spot articles described building a Tcl extension to push packets onto a network, and using the Spirent AX-4000 to characterize the rate at which packets could be sent.

We generated the IP packet by using tcpdump to sniff a packet and by copying the data into the packet-generating program. This was adequate to test that the program could work but is not versatile enough for using the packet generator to test other systems.

Modern networks use a "Russian doll" paradigm to define packets of information. A small packet is embedded in a larger packet, which is embedded in a still larger packet. And, like the dolls, each packet looks much like the others, but with subtle differences.

For example, a DNS message contains an identification field, some flags, and a payload of a set of questions and answers. This packet is enclosed in a UDP packet that includes source and destination port identifiers and a payload that consists of the DNS packet. The UDP packet gets enclosed into an IP packet, with source and destination IP addresses as identifiers and a payload of the UDP packet. And the IP packet will finally get enclosed in an Ethernet or PPP packet, with machine identifiers and a payload that consists of the IP packet.

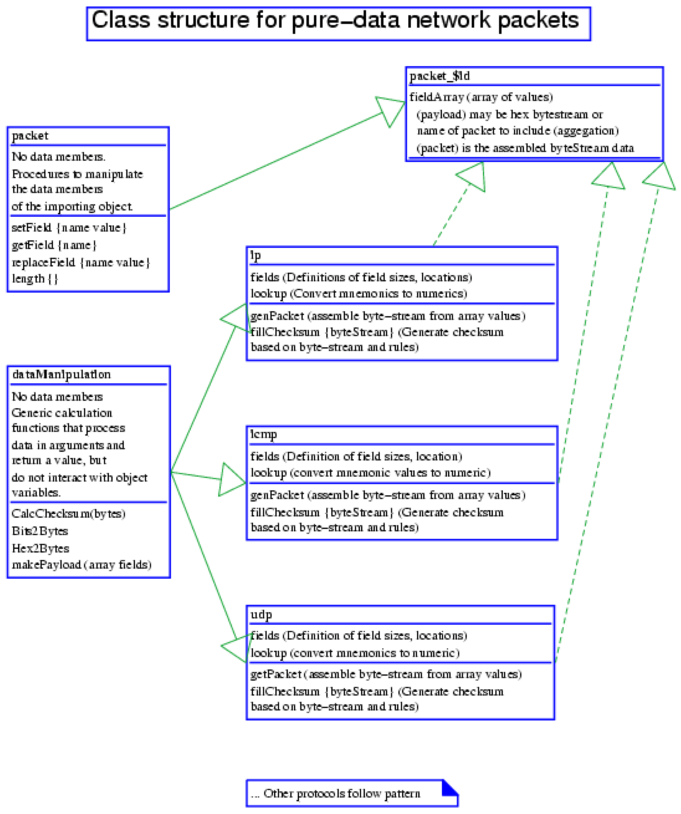
From a software design point of view, a set of similar entities with small differences is a classic situation for an object-oriented design. You can construct a base object with the core functionality and then derive special cases for the unique features.

The [incr Tcl] Tcl extension provides support for full-featured object-oriented programming, with classes, protection, inheritance, etc. For applications that require a rigorous OO implementation, this is an ideal solution. The standard Tcl distribution (ActiveTcl from *http://www.activestate.com*) includes [incr Tcl].

One of the Tcl features that makes [incr Tcl] possible is the namespace command (the [incr Tcl] extension introduced the namespace command to Tcl). The Tcl namespace provides hooks to do OO-style programming in Tcl. Using only the standard namespace command, you can implement inheritance, aggregation, and simple protection.

For this application, where I expect to have a relatively small number of object types (types of data packets – IP, ICMP, TCP, etc.) and a large number of objects (as many data packets as I need), I elected to use a very lightweight model in which the packet-type objects contain static data and methods, while the packet objects contain only data.

A Class/Object diagram of this design would resemble this:


Class structure for pure–data network packets
packet_$Id
fieldArray (array of values)
(payload) may be hex bytestream or name of packet to include (aggegation)
(packet) is the assembled byteStream data
packet
No data members.
Procedures to manipulate the data members of the importing object.
setField {name value}
getField {name}
replaceField {name value}
length {}
Ip
fields (Definitions of field sizes, locations)
lookup (Convert mnemonics to numerics)
genPacket (assemble byte–stream from array values)
fillChecksum {byteStream} (Generate checksum based on byte–stream and rules)
dataManipulation
No data members
Generic calculation functions that process data in arguments and return a value, but do not interact with object variables.
CalcChecksum(bytes)
Bits2Bytes
Hex2Bytes
makePayload (array fields)
Icmp
fields (Definition of field sizes, location)
lookup (convert mnemonic values to numeric)
genPacket (assemble byte–stream from array values)
fillChecksum {byteStream} (Generate checksum based on byte–stream and rules)
udp
fields (Definition of field sizes, locations)
lookup (convert mnemonics to numeric)
getPacket (assemble byte–stream from array values)
fillChecksum {byteStream} (Generate checksum based on byte–stream and rules)
... Other protocols follow pattern

The data objects include a single variable – an associative array indexed by the names of the fields within a packet using a particular protocol. The data is the value for that field. For example, an object named packet_1 might be an ICMP packet, which would have indices like type, code, and checksum (the minimal ICMP header fields).

One extra index is used to hold a bytestream representation of the packet, with the appropriate ordering and padding. This index is named packet.

The data objects always inherit a few basic methods from the packet class, which has no associated data, and inherit public methods and data from one *parent* class that describes the protocol.

The parent classes contain information to define the layout of fields within a packet of this type, a lookup table to convert from common mnemonics for this type of packet to a numeric value, and methods that implement the specific rules for building this type of packet. These classes invoke generic functions in the dataManipulation class as they need them.

This set of abstractions can be implemented with just the namespace command – no need for fancy OO extensions. While 700 lines of code to build ICMP, TCP, UDP, IP, and Ethernet packets is rather small, it's too much to completely discuss in this article. This article will introduce the interesting features of using the namespace command to implement this, and show how to test the output.

The Tcl namespace command provides a private area where static data and procedures can be kept. The same data and variable names can be used in multiple namespaces. Tcl namespaces are created and manipulated with the namespace command, which includes several subcommands. The namespace eval command creates new namespaces. It will evaluate a Tcl script within a namespace (and create the namespace if necessary.)

Syntax: namespace eval *namespaceID arg1 ?argN...?*

	Create a namespace, and evaluate the script arg in that scope. If more than one arg is present, the arguments are concatenated into a single script to be evaluated.
namespaceID	The identifying name for this namespace.
*arg**	The script or scripts to evaluate within namespace *namespaceID*.

This code will create a namespace that holds an associative array, and includes two functions to set and retrieve array values.

```
namespace eval packet_1 {
    variable fieldArray

    proc setField {name value} {
        variable fieldArray
        set fieldArray($name) $value
    }

    proc getField {name } {
        variable fieldArray
        return $fieldArray($name)
    }
}
```

The variable command declares that a variable exists within a namespace and may initialize the variable's value. The variables declared with the variable command are persistent and will not be destroyed when the namespace scope is exited. These variables are easily accessed by procedures within their namespace, but not from other namespaces.

Note that the syntax for the variable command is different from the global command. The variable command supports setting an initial value for a variable, while the global command does not.

Syntax: variable *varName ?value? ?varNameN? ?valueN?*

	Declare a variable to exist within the current namespace. The arguments are pairs of name and value combinations.
varName	The name of a variable.
?value?	An optional value for the variable.

The first variable fieldArray declares that the variable exists. The next variable fieldArray, inside the setField procedure, maps the fieldArray variable from the namespace scope into the local procedure scope.

The variable fieldArray is persistent, just like global variables, but exists in the packet_1 object.

Tcl namespaces are named in a tree fashion, similar to a directory tree. The separator for namespaces is a double-colon (::), and the top namespace (the default when you start up a Tcl shell) is also ::.

The first example creates a namespace named packet_1 in the scope where the command was evaluated. Assuming it is evaluated in the top-level scope, it creates the namespace ::packet_1, which contains a variable ::packet_1::fieldArray and two procedures, ::packet_1::getField and ::packet_1::setField.

Invoking the packet object's procedures by full namespace path resembles the C++/Java type of naming convention: packet_1::getField resembles packet_1-&gt;getField. This isn't really the best technique for accessing methods in Tcl. This style of invoking the procedures exposes the implementation of the object and makes the packet_1::getField invocation appear to be a unit, instead of the packet_1 being an object and getField being a method.

A Tcl-style object would be a command, and the methods would be subcommands.

Since Tcl has separate resolution tables for namespace names, variable names, and procedures, we can use the same name for a namespace, and the procedure to access it. This script will create a packet_1 command:

```
proc packet_1 {args} {namespace eval packet_1 $args}
```

When a variable named args is the last argument in a procedure definition, Tcl will assign any unassigned arguments to that variable. This allows you to define procedures that can be invoked with any number of arguments.

We can create namespaces with any script. For instance, this is equivalent to the first example:

```
set packetDef {
    variable fieldArray

    proc setField {name value} {
        variable fieldArray
        set fieldArray($name) $value
    }

    proc getField {name } {
        variable fieldArray
    return $fieldArray($name)
    }
}

namespace eval packet_1 $packetDef
proc packet_1 {args} {namespace eval packet_1 $args}
```

This makes it easy to create multiple packet namespaces, each of which includes its own copy of the fieldArray variable, which may contain unique values.

```
proc makePacket {name} {
    global packetDef
    namespace eval $name $packetDef
    proc $name {args} "namespace eval $name \$args"
}

makePacket packet_1
makePacket packet_2
makePacket packet_3
```

A downside to this technique is that the procedures are also duplicated in each namespace. While the variables in these namespaces are unique, the procedures are identical, and we don't need a new copy in each namespace. With three objects this doesn't matter, but if our application had several thousand objects, the overhead of duplicating the procedures could start to be a problem.

The namespace import and namespace export commands solve this problem (and several others we'll address later).

The namespace export command lists commands that are available to be imported. You would consider these public methods in a Java or C++ environment.

Syntax: namespace export *pattern1 ?patternN...?*

Export members of the current namespace that match the patterns. Exported procedure names can be imported into other scopes. The patterns follow glob rules.

*pattern**	Patterns that represent procedure names and data names to be exported.

The flip side is the namespace import command, which will map a procedure that exists in one namespace into the current namespace.

Syntax: namespace import *?-force? ?pattern1 patternN...?*

Imports procedure names that match a pattern.

-force	If this option is set, an import command will overwrite existing commands with new ones from the pattern namespace. Otherwise, namespace import will throw an error if a new command has the same name as an existing command.
*pattern**	The patterns to import. The pattern must include thenamespaceID of the namespace from which items are being imported.

One trick to the import command is that it maps the procedure name into the current namespace, but when the procedure is evaluated, it evaluates within the namespace it was defined in. This makes the namespace import command work well for implementing inheritance, but makes a slight problem when we try to use it naïvely to avoid duplicating procedure bodies.

The code below doesn't work.

It sets values in ::BADprocs::fieldArray, rather than ::packet_1::fieldArray or ::packet_2:fieldArray.

```
namespace eval BADprocs {
    namespace export setField getField

    proc setField {name value} {
        variable fieldArray
        set fieldArray($name) $value
    }

    proc getField {name } {
        variable fieldArray
        return $fieldArray($name)
    }
}
```

```
set packetDef {
      variable fieldArray
      namespace import ::BADprocs::*
}

namespace eval packet_1 $packetDef
namespace eval packet_2 $packetDef

proc packet_1 {args} "namespace eval packet_1 \$args"
proc packet_2 {args} "namespace eval packet_2 \$args"

packet_1 setField foo bar1
packet_2 setField foo bar2
```

The Tcl upvar command is the solution for this. Upvar will map a variable from a higher level scope into the current scope.

Syntax: upvar *?level? varName1 localName1 ?varName2? ?localName2?*

	Maps a variable from a higher variable scope into the current variable scope.
?level?	An optional level to describe the level from which the variable should be linked. This value may be a number or the # symbol followed by a number.
	The *level* defaults to 1, the level of the script that invoked the current proc.
*varName**	The name of a variable in the higher scope to link to a local variable.
*localName**	The name of a variable in the local scope. This variable can be used in this script as a local variable. Setting a new value to this variable will change the value of the variable in the other scope.

Normally, the upvar command is used to implement call-by-name (instead of Tcl's normal call-by-value paradigm). For example, a procedure to print the contents of an array would resemble this:

```
proc printArray {arrayName} {
      upvar $arrayName a
      foreach index [array names a] {
         puts "$index: $a($index)"
      }
}

array set demo {index1 val1 index2 val2}
printArray demo
```

When doing object-style programming in Tcl, we can use the upvar to map the fieldArray variable from the packet_* namespaces into the procs namespace like this:

```
namespace eval procs {
      namespace export setField getField

      proc setField {name value} {
         upvar fieldArray localArray
         set localArray($name) $value
      }

      proc getField {name } {
         upvar fieldArray localArray
         return $localArray($name)
      }
}

set packetDef {
      variable fieldArray
      namespace import ::procs::*
}

proc makePacket {name} {
      global packetDef
      namespace eval ::$name $packetDef
      proc $name {args} "namespace eval $name \$args"
}

makePacket packet_1
makePacket packet_2

packet_1 setField foo bar1
packet_2 setField foo bar2
```

Note how the namespace eval command in makePacket prepends the :: to the namespace identifier. Like file-system paths, namespace identifiers may be absolute or relative. The namespace identifier ::packet_1 defines a namespace that is a child of the global scope. The namespace identifier packet_1 defines a namespace that is a child of the current scope, which could be anything.

Creating a procedure packet_1 that uses namespace eval packet_1 setField rather than directly invoking packet_1::setField creates an entry on the procedure stack for code evaluated in the packet_1 namespace. This allows the upvar command to map the fieldArray into the ::procs namespace. If you intend to invoke methods as namespace::method, you'll need to define the methods within the namespace rather than importing them.

This technique can be generalized further to create class and new commands, just like those in Java, C++, or [incr Tcl]. Several pure Tcl object-oriented programming packages such as stooop and SNIT use similar techniques. Since this application only creates one type of object (data packets), the simple makePacket command is sufficient.

Using this technique, we can define packets that contain named fields with numeric values with code like this:

```
makePacket icmp_packet_1
icmp_packet_1 setField sourcePort 9999
icmp_packet_1 setField destPort 25
icmp_packet_1 setField sequence 1234
...
```

These values can be extracted, and assembled in the proper order, with the proper padding to create a network packet. The next trick is to define the order of the fields, sizes, etc.

In C++, Java, or [incr Tcl] the packet class would be an abstract class, and we'd derive TCP, UDP, ICMP, etc. classes from this base class with the field definitions.

In pure Tcl, we can define a namespace that contains field definitions and methods for creating network packets and import those methods into our packet object.

We commonly think of a network packet as a series of bytes. However, some fields (header length, TCP flags) are less than 8 bits long. To generalize the algorithms, I decided to build the packets as bitstreams and define the fields as bit offsets and lengths. After the bitstream is completely assembled, it's converted to bytes.

The protocol definition namespace resembles:

```
namespace eval icmp {
    # name bit-offset bit-length
    set fields {
        type 0 8
        code 8 8
        checksum 16 16 }

    # Place a bytestream representation of the packet in
    # the fieldArray associative array.
    proc fillPacket {} {
        upvar fieldArray f
        variable fields

        foreach {name start len} $fields {
            lappend bitStream [makeElement $len $f($name)]
        }

        set f(packet) [Bits2Bytes $bitStream]
    }

    # Return a bitstream padded to the required length.
    proc makeElement {len value} {}

    # Convert a bitstream to a hexadecimal bytestream.
    proc Bits2Bytes {bitStream} {}
}
```

The definition of a packet object now looks like this:

```
namespace eval icmp_packet_1 {
    variable fieldArray      ;# Array of values for fields

    namespace import ::proc::*
    namespace import ::icmp::*
}
```

A bytestream packet can be constructed with code like:

```
icmp_packet_1 setField type 0
icmp_packet_1 setField code 0
icmp_packet_1 setField checksum 0
icmp_packet_1 fillPacket
```

The calls to setField will be evaluated in the ::procs namespace, which will use upvar to map the ::icmp_packet_1::fieldArray variable into the local procedure scope.

The call to fillPacket will be evaluated in the ::icmp namespace and will use upvar to map the ::icmp_packet_1::fieldArray variable into this scope. The ::icmp::fields variable is already in the correct scope.

By defining different values in the fields variable, we can define different protocols. These descriptions can be used to create different types of data packets by importing the appropriate namespace.

This leads to a more generalized makePacket procedure:

```
proc makePacket {name type} {
    global packetDef
    namespace eval ::$name $packetDef
    namespace eval ::$name "namespace import ::${type}::*"
    proc $name {args} "namespace eval $name \$args"
}

makePacket icmp_packet_1 icmp
```

The final useful tweak to the makePacket procedure is to support setting the fieldArray values when the object is created, instead of creating an empty object and requiring multiple setField calls.

The array set command assigns values to multiple associative array indices in a single command. For example, this command would assign 0s to fieldArray(type), fieldArray(code), and fieldArray(checksum):

```
array set fieldArray {type 0 code 0 checksum 0}
```

This makePacket procedure definition creates the new object, populates the fieldArray variable, inherits the field definition and packet-generating methods, and creates the procedure to use for further interaction with the new object.

```
proc makePacket {name type args} {
    global packetDef
    namespace eval ::$name $packetDef
    namespace eval ::$id [list array set fieldArray $args]
    namespace eval ::$name "namespace import ::${type}::*"
    proc $name {args} "namespace eval $name \$args"
}
```

```
# ...
makePacket icmp_packet_1 icmp type 0 code 0 checksum 0
```

In actual fact, this package is a bit larger and more complex than described. It allows values to be defined with mnemonics as well as numbers, translates from Internet and Ethernet style addresses into bytestreams, checks for a complete set of values, etc. The use of namespaces to implement an abstract class and lightweight objects, however, works as described.

In the actual packet-generating package, the makePacket command is named make and is contained in the ::packet:: namespace.

The first thing to do with the network packet package is to confirm that it generates the expected packets. There are more ways to test software than there are software writers. In this case, the tests break down into two categories:

1. Tests that compare the packet data to a known good bytestream
2. Tests that use external systems to analyze packets "off the wire"

The Tcl interpreter comes with a package for automating regression tests. This package is used to test the Tcl interpreter (and many other Tcl packages) and to invoke make tests. The two workhorses of this suite are:

Syntax: tcltest::test *name desc constraint script expectedAnswer*

	Run a test, and compare the results to the expected results.
name	The name of this test – to use when reporting pass/fail results.
desc	The description of this test – to use when reporting pass/fail results.
constraint	A set of constraints – to define when this test should be evaluated. (For example, only test on certain platforms, if other tests pass, etc.)
script	The test script to evaluate.
expectedAnswer	The expected result.

A simple test would resemble:

```
tcltest::test expr-1 "Confirm that expr will add 2+2" {} \
      {expr 2+2} 4
```

If the test fails (for instance, if we declare a wrong value for the expectedAnswer), Tcl generates output resembling this:

```
==== expr-1 Confirm that expr will add 2+2 FAILED
==== Contents of test case:
expr 2+2
—— Result was:
4
—— Result should have been (exact matching):
5
==== expr-1 FAILED
```

After running the tests, we can generate a report with the ::tcltest::cleanupTests command. The ::tcltest::cleanupTests command generates a report resembling this:

```
:     Total 2     Passed  1     Skipped 0     Failed  1
```

To test the packet-generating package, we can generate the expected packets by hand and compare these values to the output from the code.

A set of tests like this can exercise a package and confirm that it behaves as expected:

```
package require tcltest
package require packet

set p1 [packet::make ICMP type ICMP_ADDRESS code 0 \
      checksum 0 identifier 1 sequence 2 subnet 00]

set expected [list 0x11 0x00 0xee 0xfc 0x00 \
      0x01 0x00 0x02 0x00 0x00 0x00 0x00]

tcltest::test makeICMP-1 {ICMP_ADDRESS} {} \
      {$p1 getField packet} $expected

# ... more tests
::tcltest::cleanupTests
```

This produces a regression suite that runs quickly, provides a nice summary of results, and is easy to create. The downside is that it confirms that the package does what I expect, which may not be what's correct.

This is where an outside validation tool is useful. One easily available tool for this testing is tcpdump. We can generate packets, transmit them, and let tcpdump sniff the packets, do some analysis, and report problems.

For instance, the tcpdump command

```
tcpdump -s 15000 -l -x -n -v -i eth1
```

will generate output like this:

```
14:22:12.963048 192.168.9.2 &gt; 192.168.9.17: icmp:
      address mask request (ttl 32, id 2, len 32)
          4500 0020 0002 0000 2001 0778 c0a8 0902
          c0a8 0911 1100 eefc 0001 0002 0000 0000
14:22:12.963051 192.168.9.2 &gt; 192.168.9.17: icmp:
      address mask request (wrong icmp csum) (ttl 32, id 2, len 32)
          4500 0020 0002 0000 2001 0778 c0a8 0902
          c0a8 0911 1100 ee99 0001 0002 0000 0000
```

When testing packets generated with this code:

```
lappend auto_path .

package require packet
package require ip
package require icmp
package require ether
package require tcp
package require udp
package require dnetlib

load ./libdnet.so

set e [dnet::open eth1]

# Generate a good icmp Address Mask Request.
set icmp1 [packet::make ICMP type ICMP_ADDRESS code 0 checksum 0 \
    identifier 1 sequence 2 subnet 00]

# Generate a BAD CHECKSUM icmp Address Mask Request.
set icmp2 [packet::make ICMP type ICMP_ADDRESS code 0 checksum 99 \
    identifier 1 sequence 2 subnet 00]

# Embed the icmp packets into IP packets.
set ip1 [packet::make IP version 4 hdrlen 5 tos 0 length 32 id 2 flag 0 \
    offset 0 ttl 32 protocol ICMP checksum 0 source \
    192.168.9.2 dest 192.168.9.17 options {} payload [$icmp1 getField packet]]

set ip2 [packet::make IP version 4 hdrlen 5 tos 0 length 32 id 2 flag 0 \
    offset 0 ttl 32 protocol ICMP checksum 0 source \
    192.168.9.2 dest 192.168.9.17 options {} payload [$icmp2 getField packet]]

# Embed the IP packets into Ethernet packets.
set ep1 [packet::make ETHER dest 00:E0:4C:00:14:4D src 00:A0:CC:D1:B6:00 \
    type IP payload [$ip1 getField packet]]
set ep2 [packet::make ETHER dest 00:E0:4C:00:14:4D src 00:A0:CC:D1:B6:00 \
    type IP payload [$ip2 getField packet]]

# Convert the Ethernet packets into binary.
dnet::importPacket $ep1
dnet::importPacket $ep2

# Send the binary packets out into the world.
dnet::send $e $ep1
dnet::send $e $ep2
```

These two sets of tests (one internal, and one external) are probably adequate. But proper testing should use the most expensive and complex piece of equipment you can access. After all, how else can you justify nifty toys?

The Spirent AX-4000 was described briefly in the previous Tclsh Spot article. The next article will discuss using the AX-4000 to capture and analyze the output of the packet generator.

practical Perl
fixing broken modules

by Adam Turoff

Adam is a consultant who specializes in using Perl to manage big data. He is a long- time Perl Monger, a technical editor for *The Perl Review*, and a frequent presenter at Perl conferences.

ziggy@panix.com

Introduction

I recently worked on a project where I needed to make some bug fixes to some locally written Perl modules. To make my changes, I fixed and tested a local copy of these modules, modifying Perl's search path to find my copy of them instead of the buggy versions. Modifying Perl's search path is an excellent way to test experimental code without the hassle of installing each fix, or impacting other users on the same system.

Dynamic languages like Perl, Python, and Ruby all have one very important feature in common: Programs written in these languages are distributed in source form. This is a great boon to software developers who need to debug a system after it is installed. A Perl programmer can examine a large Perl application and see exactly how it works and even make changes, if necessary. Debugging an installed application written in Perl is trivial. Debugging a program written in C, C++, or Java is more difficult, especially if the original source code is lost, misplaced, or otherwise unavailable.

Access to a program's source code eases repair. Once I find a bug in a Perl program that needs fixing, I can easily copy the program, make a change, and test the updated program. I can then replace the original program with my fixed version if I have write access to the appropriate directories. If I do not have sufficient permissions to upgrade the program, I can maintain the updated program in a local portion of my path while I wait for installation issues to be sorted out. In the worst case, I can update my PATH environment variable to find my updated program before the existing version.

Fixing a broken library module is a similar process, but slightly more complicated. If I do not have write permission to the library module directories, then I cannot install the update. Even if I could, there are good reasons not to update a module that could be used by many programs and users on a system. It is entirely possible that my blindly overwriting an existing module with my updated version will fix my program, but it will also break a great many other programs in use.

If I want to install an updated module locally, I need to update Perl's module search path to find my update before the preexisting version. Because this is Perl, there's more than one way to do it. Which technique I will use will depend on the situation.

Perl's Module Search Path

Perl processes programs in two phases: compile time and runtime. During compile time, Perl ensures that your program is syntactically correct and performs other operations, like loading modules. Once this process is complete, the runtime phase begins and Perl starts to execute your program statements.

During compile time, Perl includes external modules by looking through a list of directories named by @INC until it finds an appropriate file to load. To include modules from a specific directory, update @INC during compile time and before use statements are processed. Modifying @INC at runtime will have no impact since the program has been compiled and all use statements have already been processed.

By default, Perl looks in one of two general file-system areas when a module is to be loaded. Core modules (the ones that are bundled with Perl) are stored in $PREFIX/lib/perl5, where $PREFIX is the base of the Perl installation, like /usr, /usr/local, or /opt. Additional modules, like those installed from CPAN, are stored in $PREFIX/lib/site_perl. These directories may also include subdirectories containing Perl's version number (e.g., 5.005_03, 5.6.1, or 5.8.0) and subdirectories containing the current platform architecture (i386-freebsd, darwin, etc.).

If a module is not found in any of these locations, Perl will look in the current directory. If Perl still cannot find a module, it will terminate processing the program and report a fatal error.

Whenever Perl is loading a module, it looks for the first matching module encountered in the list that is the search path. If I want to override a previously installed module, I must place it in a directory that will appear before the normal module search directories.

Updating the Module Search Path

There are many ways to update the module search path. One way is to use the PERL5LIB environment variable. Using environment variables to change Perl's behavior is generally discouraged, because "opaque" settings are hard to notice, especially by a casual maintainer; besides, they can vary on a per-user or per-terminal basis. Sometimes, setting PERL5LIB is the best way to update the module search path, like communicating with Perl subprocesses.

PERL5LIB adds new library directories to the front of the module search path. It can contain a series of colon-delimited directories:

```
[ziggy@duvel ~]$ mkdir newlib1 newlib2
[ziggy@duvel ~]$ export PERL5LIB=newlib1:newlib2
[ziggy@duvel ~]$ perl -le 'print join("\n", @INC)' newlib1
newlib2
/opt/lib/5.8.0/darwin
/opt/lib/5.8.0
/opt/lib/site_perl/5.8.0/darwin
/opt/lib/site_perl/5.8.0
/opt/lib/site_perl
```

Whenever a directory in PERL5LIB contains a subdirectory that matches the current platform architecture, that platform-specific directory will also be added to the search path. This is also the location where compiled C extensions will be installed.

```
[ziggy@duvel ~]$ mkdir newlib2/darwin
[ziggy@duvel ~]$ export PERL5LIB=newlib1:newlib2
[ziggy@duvel ~]$ perl -le 'print join("\n", @INC)' newlib1
newlib2/darwin
newlib2
/opt/lib/5.8.0/darwin
/opt/lib/5.8.0
/opt/lib/site_perl/5.8.0/darwin
/opt/lib/site_perl/5.8.0
/opt/lib/site_perl
```

Another way to extend the module search path is to use Perl's -I command line switch. Adding multiple -I switches when invoking Perl will add multiple directories (and version-specific subdirectories) to @INC. Note that adding -I switches on the command line will prepend directories to @INC, while using -I on the shebang (#!) line will append directories to the end of @INC:

```
[ziggy@duvel ~]$ cat > test.pl
#!/usr/bin/perl -lw -Inewlib2
print join("\n", @INC);
^D
[ziggy@duvel ~]$ perl -Inewlib1 test.pl
newlib1
/opt/lib/5.8.0/darwin
/opt/lib/5.8.0
/opt/lib/site_perl/5.8.0/darwin
/opt/lib/site_perl/5.8.0
/opt/lib/site_perl
.
newlib2/darwin
newlib2
[ziggy@duvel ~]$
```

With this behavior, using -I on the shebang line is sufficient for adding a module directory to @INC to find modules that are not stored in the core or site module directories. If you want to include a directory in @INC to supersede the modules installed elsewhere on the system, you must specify -I on the command line when invoking Perl.

A third way to add directories to @INC is to modify @INC directly at compile time. One way to do this is to modify @INC in a BEGIN block, so that it will be modified before modules are loaded:

```
#!/usr/bin/perl -lw
BEGIN {unshift(@INC, "newlib1", "newlib2"); } print
join("\n", @INC);
```

No output is produced.

Using BEGIN blocks and directly tweaking @INC works, but it is ugly and obscure. A better way to perform the same task is to use a use lib; declaration instead. This declaration is processed at compile time, before modules are loaded. The use lib; declaration does exactly what it says — prepends another library path to the list of module search paths.

```
[ziggy@duvel ~]$ perl -lw
use lib qw(newlib1 newlib2);
print join("\n", @INC);
^D
newlib1
newlib2
/opt/lib/5.8.0/darwin
/opt/lib/5.8.0
/opt/lib/site_perl/5.8.0/darwin
/opt/lib/site_perl/5.8.0
/opt/lib/site_perl
.
```

Note that use lib; declarations can insert directories at the front of @INC, but each directory must be explicitly added. Only PERL5LIB and perl -I can automatically add a platform-specific subdirectory to @INC to find compiled modules.

Using Local Module Directories

Updating the module search path has a great many uses. The most common use is to load modules from an application-specific library directory. Installing modules in a local library makes it easy to quickly modify an application's modules when it is in development. This eliminates the need to go through the module-install process for each update.

If you cannot or do not want to add modules to the standard module library, you can install modules elsewhere and find them with a use lib; declaration. This is a great way to test modules before installing them, or install modules in a central location on a system where you cannot install a module in the normal site-wide location.

Fixing a Broken Module

One of the lesser-known ways to use a local library directory is to fix a broken module. Because I can create a local module library directory and configure Perl to look there for modules, I can make a copy of a broken module and fix it. I can update programs to look for this fixed module, or invoke Perl using PERL5LIB or perl -I to find my fixed module.

Consider this little module, which misbehaves and emits an obnoxious number of status messages:

```
package Sample;
use strict;
$|++;   ## Turn on auto flushing
sub new {
      print STDOUT "Creating a new object\n";
      my $object = {};
      bless $object, __PACKAGE__;
      $object->load_config();
      return $object;
}
sub load_config {
      print STDOUT "Loading configuration file\n";
                      my %config;
      $/ = "\n";   ## Read in a series of lines
      open(F, "/etc/Sample.conf");
      while (<F>) {
          chomp;
          s/#.*$//;
          s/s+/ /;
          next unless m/^(.*?)=(.*)$/;
          $config{$1} = $2;
      }
      return %config;
}
...
1;
```

This module contains a few errors I want to fix. I start by copying Sample.pm from its location in /opt/lib/perl5/5.8.0 to my local module directory, ~/fixed-modules/. I then make my changes to ~/fixed-modules/Sample.pm.

In some cases, I could fix this module by writing a new module and inheriting from Sample.pm. However, the problems that I want to fix are endemic and cannot be fixed effectively without rewriting the entire module.

Another approach would be to create a fix to this module and rename it as FixedSample.pm. If I have several programs that use the Sample module, then I will have a lot of programs to update to use FixedSample instead. By making a fixed version of Sample available, I can continue to use existing programs with little or no modifications to those programs.

The first error I need to fix is the modification of the $| special variable at the beginning of the module. This global variable controls "auto flushing," or automatically flushing data sent to STDOUT instead of buffering it. Modification to $| is usually a global change in program behavior. Methods in this module may want to have output to STDOUT flushed immediately, but other portions of this program may rely on STDOUT being buffered. Changing global behavior like this is bad style when writing a module.

A better way to flush output to STDOUT within a module is to modify the value of $| locally. This can be done by making a local copy of $| within a sub and modifying that copy:

```
sub new {
      ## Turn on autoflushing only within this sub
      local $| = 1;
      print STDOUT "Creating a new object\n";
      ...
}
```

There is a similar problem in load_config(). It modifies the $/, or input record separator. Within load_config(), this variable needs to be a newline character (the default value). Another portion of the program may need a different behavior, like reading in an entire file at once, or reading in one block at a time. These behaviors are defined by setting $/ to undef or the empty string. Calling load_config() will change this global behavior and may inadvertently impact other portions of the program.

Fixing this buggy behavior also requires making a local modification to the value of $/:

```
sub load_config {
      ## turn on autoflushing locally
      local $| = 1;
      print STDOUT "Loading configuration file\n";
                      my %config;
      ## Read in a series of lines within this sub
      local $/ = "\n";
      ...
}
```

Another problem I want to fix deals with opening the configuration file. Remember that file handles in Perl are global variables. The configuration file is opened using the file handle fh, which is a very common name for a file handle. If my program already has an open file handle called fh, then load_config() will close it and open up another file instead. I could fix this problem by choosing a better name for my file handle. A better solution would be to use a lexical file handle, or a file handle that exists only within this sub:

```
sub load_config {
      ...
      open (my $fh, "/etc/Sample.conf");
      while (<$fh>) {
          ...
      }
      ...
}
```

After I make these changes, my copy of the Sample module will be well behaved. But it still emits an obnoxious number of log messages. I don't need to see them, and I would like to eliminate them. Here is what my fixed version of this module looks like after I've removed the unnecessary print statements:

```
package Sample;
use strict;
sub new {
      my $object = {};
      bless $object, __PACKAGE__;
      $object->load_config();
      return $object;
}
sub load_config {
      my %config;
      local $/ = "\n";
      open(my $fh, "/etc/Sample.conf");
      while (<$fh>) {
          chomp;
          s/#.*$//;
          s/s+/ /;
          next unless m/^(.*?)=(.*)$/;
          $config{$1} = $2;
      }
      return %config;
}
...
1;
```

With an updated version of my module, all I need to do now is use it in my programs. I can do this in a number of ways. I can modify programs that I explicitly want to use this module by adding a use lib '$ENV{HOME}/fixed-modules';. If I do not want to modify my Perl programs, I can use perl -I~/fixed-modules when invoking programs that use this module, or set PERL5LIB to include ~/fixed-modules. Any of these techniques will allow me to override Sample.pm with my copy.

Annotating Unfamiliar Modules

Periodically, I need to fix a module I have never seen before. I might be able to isolate a problem using the Perl debugger, but any insights I gain will probably be lost by the time I finish a debugging session. I could write them down on paper or online somewhere, but there is no guarantee I'll have that paper or file available when I need it again. Usually, it will be sitting on my desk at home when I am at work, or sitting on my desk at work when I am at home.

A better solution would be to comment on the program's source code directly. In this situation, I probably do not want to update the installed module directly, but I can comment on a module if I create a local copy and annotate that copy. If my changes are useful, then I can easily create a patch to send back to the module's author.

Another modification I can make locally is to normalize a module's coding style. Many module suites are written by multiple programmers over a period of time. Sometimes the style of each author will vary, or the coding style is so different from my own that it makes the module difficult to read.

I'd rather fix a problem than complain about coding style. If I make a local copy of a module, I can process it using a code formatter like perltidy and apply a single, readable style to one or more modules. My goal here is to understand how a module works, not to start a flame war over style. By loading my reformatted version of the modules, I can also use them when debugging a program. Once I understand how the module works, I can patch the original version of the module.

Conclusion

Because Perl programs and Perl modules are distributed as source code, it is easy to take an existing program and modify it to quickly fix a bug. This kind of fast turnaround helps you solve a problem before the boss fires you.

Fixing modules takes a little more effort than fixing a broken program, but it can be done. The key to fixing a broken module rests with loading modules from a local directory, and updating Perl's module search path, @INC, to find the updated modules.

musings

by Rik Farrow

Rik Farrow provides UNIX and Internet security consulting and training. He is the author of *UNIX System Security* and *System Administrator's Guide to System V*.

rik@spirit.com

Perhaps the mystery of the trojaned open source code has been solved. In my February column, I discussed the trojaned configure scripts that would connect to a fixed IP address, and exec a shell if there was any response. On January 22, CERT published an advisory (*http://www.cert.org/advisories/CA-2003-02.html*) about a double free() bug in CVS servers that allowed an attacker to execute code as root, requiring no more than read-only access to start with.

We have no way of knowing if this will be the only bug ever discovered in CVS or on other services that run on mirrors used to distribute open source software. But it sure makes having signed distributions seem a lot more important. Of course, that precludes having enough signers on your own key ring that you can trust the public key used to sign the code.

Other great events have occurred. The world's fastest spreading and shortest lived worm appeared about 0530 UTC on January 25, late Friday night in the US, where it appeared to have been launched. Sapphire, or Slapper, attacked unpatched versions of Microsoft SQL Server and MSDE. As usual, a patch had been published by Microsoft months before the worm was released. But even Microsoft got the infection, and was still fighting it on Monday morning. Bank of America and Imperial Bank of Commerce ATMs went down on Saturday, and some other well-known organizations (AMEX, A.C. Nielsen, Price Waterhouse, and Citicorp) were hit hard as well.

Perhaps these organizations should not be blamed for not patching their Windows boxes, as Microsoft had put out a later hot fix for MS SQL that would have regressed the DLL involved, ssnetlib.dll, after the initial security patch was posted. Or perhaps the sysadmins involved had no idea they had an open network service listening at a UDP port attached to MS SQL Server. Keep in mind that many applications embedded MSDE, a developer-only version of MS SQL, so that someone running McAfee Centralized Virus Admin, Veritas Backup Exec, or ISS RealSecure 7.0 and Scanner were also vulnerable. ISS, a security company, included a vulnerable version of MSDE at least as late as September 2002, two months after the patch had been announced, and never reported that their own software was vulnerable to the attack.

Sapphire was the fastest spreading worm yet seen. A paper describing Sapphire's spread (*http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html*) explains why. In brief, Sapphire was fast because it used short UDP packets (404 bytes, including headers) to attack, so no TCP handshaking, which caused Code Red to spread much more slowly. The number of Sapphire-infected systems doubled every 8.5 seconds, reaching 90% of all vulnerable hosts in 10 minutes. If you had read the Paxton paper presented at last summer's USENIX Security Conference (*http://www.icir.org/vern/papers/cdc-usenix-sec02/*), Sapphire might not have seemed such a surprise.

The main effect of Sapphire was denial of service. Some networks became unreachable within minutes of the start of the attack. But many ISPs acted quickly to block packets going to port 1434/UDP, which is, after all, not a required or necessary service. The effects on the greater Internet were soon damped down, and almost back to normal within 14 hours: *http://www.matrixnetsystems.com/ea/2003/20030125_packetloss.jsp*.

I found Sapphire interesting because it so easily penetrated "protected" networks. No firewall maintainer in his or her right mind allows arbitrary UDP packets to enter from the Internet. So how did Sapphire get inside financial institutions, Microsoft, and many other sites? The answer includes VPNs, connections to home systems, and various routing leaks.

Books

The notion of various undocumented connections to networks reminds me of one of the books I have been reading lately. The venerable *Firewalls and Internet Security* (Cheswick, Bellovin, and Rubin, Addison-Wesley, 2003, ISBN 0-201-6346-6) has finally come out in a second edition. I had quit bugging Ches about this, feeling it was a futile gesture, but am glad to see the project completed at last.

Firewalls used to be my favorite security book, and still comes close (it now has to compete with Ross Anderson's *Security Engineering*). The first edition (from 1994) didn't even mention HTTP, so the book really needed revision. The new book contains new chapters, including a short one on Web services, IDS, and Network Layout, as well as updates to most other materials. Occasionally, I would find something that seemed very out-of-date, like the mention on page 58 that most sniffers are discovered when a disk fills up or that filtering with routers entails a performance penalty, but that the Internet is "connected by (at best) a DS1 (T1) line (1.544 Mb/sec)."

The writing in *Firewalls* is terse. You will find information about the most popular network services and the issues involved in passing them through firewalls, as well as the related risks. An "evening with Berferd" survives, and there is also a new section about a different attack, complete with crude forensics used to examine Clark, the ULTRIX system in question. While I wouldn't recommend this book to firewall "newbies" (I *don't* want them attempting to rip out the IP forwarding routines in their kernel), I do recommend it to people who must run firewalls and who already understand the difference between an application gateway and a circuit relay.

Matt Bishop also has a new book out: *Computer Security, Art and Science* (Addison-Wesley, 2003, ISBN 0-201-44099-7). Matt wrote his book as a textbook for advanced undergraduates and graduate students of computer science. In other words, you can learn a lot about security using this book in addition to using it to teach security, but it is not a book for people wanting to improve the way they maintain system or network security. Perhaps getting Matt to teach a couple of semesters to the programmers who write our network servers and authentication code might be a good idea, though. And if your goal is to really learn about computer security, Matt's book is clear, easy to read, thorough, and includes exercises as well as telling you which chapters to skip if your math skills are rusty (or perhaps never at the level of following formal proofs).

I was fortunate enough to get to tech edit Mick Bauer's *Building Secure Servers with Linux* (O'Reilly, 2002, ISBN 0-596-00217-3). That forced me to read his book in excruciating detail, which he later claimed he appreciated. *Building* explains securing a Linux server at a very practical level. There is a section that talks about risk analysis at the beginning, but then the book gets very nuts-and-bolts, with examples of firewalling, OS hardening, using SSH, OpenSSL, securing DNS, SMTP, Web servers, FTP servers, handling logs, and some simple IDS techniques.

I didn't just read Mick's book, I used his instructions to do things that I had put off, like putting BIND in jail. Note that I did this on a BSDi box, not a Linux box. I believe that most of the configuration examples will work just as well on any OS where a server package is supported (except Windows), so this book can have a broader audience than just Linux users. If you find you need something that goes beyond what comes as documentation for some open source security package, check out Mick's book. The writing is clear, it's well organized, and provides enough detail for you to install and configure the services it covers.

And, finally, you can read a book about *BGP* (van Beijnum, O'Reilly, 2002, ISBN 0-596-00254-8). You might wonder why I would read a book about BGP besides curiosity, but there are important intersections between BGP and security. Van Beijnum does a good job explaining issues like peering and how that affects a site that wants or needs multiple connections to the Internet for reliability. Filtering is also vitally important to the proper functioning of BGP (and in being a good neighbor) and is covered, but not in its own chapter.

The Finale

The Internet survived yet another attack. While Sapphire disrupted local connectivity, and access to many sites became difficult, the Internet as a whole continued running. And the quick efforts to dampen the packet flooding caused by Sapphire, estimated at 55 million packets per second at its peak (CAIDA paper), were accomplished by the night-shift personnel at ISPs, not by government organizations.

Sapphire could have been much worse, though not in the sense that it could have killed or maimed people. In a bit of cheerful news in an otherwise depressing time, Marcus Sachs (director of communications infrastructure protection in the Office of Cyberspace Security in Washington, DC) said in a response to Sapphire, "There was no lasting damage done to the infrastructure. We'd like to see the term cyber-terror dropped." Yes, someone in the US government who understands that terrorism involves "something that physically kills people" (his words).

But all Sapphire did was invade and attack. The SQL module involved, ssnetlib.dll, runs with Local System privileges. Had the author been a bit less frugal with his or her code, he or she could have included a timer that stopped the worm from scanning about 15 minutes after it was released. A second timer could then have counted down the seconds until some other activity began, whether it was a different DoS (like Code Red flooding the IP address for whitehouse.gov) or perhaps something a bit nastier, not as subtle as disk formatting, but some other form of data destruction. Or Sapphire could have downloaded and installed DDoS agents. A relatively unheard-of trojan named Leaves had been installed on over 20,000 systems last summer and communicated via IRC. The author of Leaves, a British man, was caught before he used his network of agents for anything.

The Internet has proven to be remarkably resilient. The security of Internet-connected systems, on the other hand, has been proven to be a rare beast, although it probably does exist in a minority of locations. Perhaps more people need to follow the advice of Cheswick, Bellovin, and Rubin (install bulkhead firewalls internally) and of Mick Bauer (harden those servers!).

hacking for fun and profit

As terms borrowed from classic American westerns, often inhabited by black-hatted villains and white-hatted heros, a "black hat" cracker describes someone who breaks into a computer system or network with malicious intent; a "white hat" is a cracker who identifies a security weakness in a computer system or network so that the system's owners can fix the breach before it is exploited. White-hat cracking is a hobby for some while others provide their services for a fee. The paid white-hat cracker may work as a consultant or be a permanent employee on a company's payroll.

by Tina Darmohray

Tina Darmohray, contributing editor of *;login:*, is a computer security and networking consultant. She was a founding member of SAGE. She is currently a Director of USENIX.

<tmd@usenix.org>

Regardless of hat color, both kinds of crackers are trying to achieve the same goal: successfully breaking into networks and computers. When used by the white hats, these targeted break-in attempts are often part of an overall security plan and are referred to as penetration tests. While you never know what you're going to find when embarking on such a test, it turns out that what they are (and are not) good for is well understood by security professionals. I asked over a dozen such individuals when they'd recommend penetration tests for an organization and got a lot of feedback, with a few variations on the theme. So if you're wondering what penetration testing can do for you, read on.

First off, penetration testing is part of an overall security plan. Security testing, like many other pieces of a total security solution, is not going to provide security to your site all by itself. And, in fact, penetration testing is really a post-implementation verification tool rather than something like a firewall or VPN concentrator, which are actually implementing part of the security at a site. There's a lot of legwork to be done before any verification is begun. Appropriate use of a penetration test implies that the up-front work is complete and is ready to be tested. The professionals reiterate that penetration testing is not a replacement for careful security design and implementation and that these designs begin with a thorough risk assessment and management buy-in of the organization's IT security priorities.

Once the security goals have been hammered out and a solution is in place, it's time to haul out the white hats and verify you've hit your target. Penetration testing is recommended to verify a site's initial security deployment and as a follow-up whenever new systems are deployed or existing systems are reloaded, upgraded, reconfigured, or patched, or when there are code changes to exposed services or applications. Thereafter, on a semiannual or annual basis, penetration tests are recommended to ensure that you're maintaining the security stance you've adopted and that your exposure profile has not degraded. Major vulnerabilities that lead to remote privileged access come out all of the time, so even though you passed the last test with flying colors, it helps to have one done at regular intervals. One frequent recommendation is to consider hiring someone else to do some of these follow-up tests, just to get a fresh set of eyes looking at your site.

Whether a penetration test is hired out or performed in-house, the tools and methodology used can determine the true value of the test results. Simply pointing a scanner at a site is not penetration testing; there must be a diligent effort to look at the site from several different angles, using a myriad of tools and some hand inspection of custom code and components. Better testers continue until they have found at least one vulnerability. The best testers carry out a test plan that includes a list of planned tests, so that they test much more than just the firewall or public servers, but look for other ways into the site.

Penetration tests are the technological equivalent of having the manager of the local bank, before going home, walking about to ensure that all of the windows, doors, and safes are shut.

Aside from straightforward verification, penetration tests appear to be used for a variety of other, perhaps more political, reasons. Some organizations use them as "feel good" reassurance which they provide to third parties to prove a certain level of security is in place. Sadly, sometimes it's necessary to use them to assist in in-house battles: "Hey! You can't put that application on the Web server; it's a huge vulnerability!" "Well, it's too late, we've gone live with it, and we won't shut it off unless you prove there's a problem." You know what happens next . . .

Prevailing practice indicates, however, that penetration tests have their place in the standard security suite, with the proviso that most of the security risk is still from authorized users. This is the old branch bank scenario. Penetration tests are the technological equivalent of having the manager of the local bank, before going home, walking about to ensure that all of the windows, doors, and safes are shut. This will not prevent the assistant manager from giving the key to an in-law who then breaks in on Sunday night. Similarly, a penetration test will not stop the technological equivalent of the assisted in-law, but the embarrassment factor and concomitant loss of business to a bank that loses money because the window was open is much greater than the errant in-law loss. That is true even if the in-law losses are larger than the loss as a function of the open window. It is the appearance that the basics weren't done right that is most damaging.

One last thing to keep in mind with regard to penetration testing: If it isn't successful, that doesn't mean you're safe, just that the right thing wasn't tried. It's the old problem of trying to prove the negative, and whether you do it yourself or hire it out, there is only so much time.

ups and downs of UNIX/Linux host-based security solutions

by Anton Chuvakin

Anton Chuvakin is a senior security analyst with a major information security company. His areas of infosec expertise include intrusion detection, UNIX security, forensics, and honeypots. In his spare time he maintains his security portal at *http://www.info-secure.org*.

anton@netForensics.com

An exciting area of UNIX, security presents many challenges not resolved by a single technology. Host-based security solutions occupy a crucial place between network-based defenses (e.g., firewalls and network intrusion detection) and good administrative practices (e.g., securely configured and hardened hosts). Many of the SANS Top 20 Vulnerabilities can be successfully mitigated using host-based security. For example, buffer overflows in Sendmail, BIND, and RPC services can be secured using kernel-call tracing, and the consequences of most of the other attacks from the list can be discovered using file-system integrity checking.

Host-Based Security Technologies

This paper deals with UNIX host-based intrusion detection and prevention. We will take a look at UNIX host security solutions, with the focus on their inadequacies and ways to overcome them (whenever possible). Currently, many different technologies – integrity checking, kernel- and system-call tracing, log analysis, local-host NIDS – are lumped together into host-based security.

Integrity-checking software examples include Tripwire, AIDE, and a large number of lesser-known scripts and applications. Linux RPM also provides some integrity-checking functionality ("rpm -V" mode). The common feature of integrity checkers is that they keep a record of file properties such as modification or change times, location on disk, permissions, owner, and other attributes. They also compute and keep cryptographic checksums of the file contents. The simplest application of this kind would be a shell script similar to the following:

```
#!/bin/sh
#primitive integrity checker

ls -laRi / >/home/files
cd /usr/bin
ls | xargs -i md5sum {} > /home/sums

diff /home/files /home/files.old
diff /home/sums /home/sums.old
```

HIDS such as Entercept operate on a kernel level. While high-security kernel patches (such as Linux LIDS and Solaris Pitbull) also work in kernel space, they are more accurately described as "prevention" technology as opposed to "detection," since they restrict the actions of users and applications based on certain pre-defined access control lists. Some Linux kernel modules (such as StMichael and StJude – *http://sourceforge.net/projects/stjude*) occupy an intermediate space: They can detect malicious kernel modules and can also prevent some of the damage caused by them.

The main distinction to be found in UNIX system log analysis tools is between real-time tools (that read log records as soon as they are produced, i.e., written to disk) and

Host IDS usually presents a bigger administrative and management challenge than network intrusion detection.

cron-based tools (that run periodically and process accumulated log records). Some commercial HIDS (such as Dragon Squire) also have capabilities to analyze system logs. In addition, there are many freeware tools that can be used to detect intrusions using log files. Swatch is the best-known real-time tool, while logcheck, logwatch, and many others provide periodic log assessment.

Network IDSes that are used to monitor traffic only destined for a specific host (also sometimes called hybrid IDS) will not be considered here since they are much closer to network intrusion detection systems than to host security. However, if such a "hybrid" IDS analyzes network traffic after processing by the host protocol stack (e.g., at the application level), it can do many things that normal network IDS cannot do, such as analyze encrypted traffic (SSH, SSL, IPSec VPN, etc.). In addition, many of the insertion and evasion attacks described in the Ptacek and Newsham paper ("Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection") will not work against such IDS, since raw traffic processing is done by the target system stack and not by some other host (with its own network stack peculiarities) sniffing the traffic.

HIDS Challenges

What are some of the advantages of host-based intrusion detection products? The key difference is that while network IDS detects potential attacks (which are being sent to the target), host IDS detects attacks that succeeded, thus having a lower false positive rate. Although some might say that network IDS is thus more "proactive," host IDS is effective in the switched, encrypted, and high-traffic environments that present certain difficulties to NIDS.

Now it's time to turn to HIDS challenges.

First, at least some of the host IDS components are deployed on an attacked host. If the attack succeeds, the intruder will usually have "root" access to the box and might be able to disable or deceive the IDS. The usual countermeasures are "hide" (prevent an intruder from seeing the IDS and continue operation), "run" (sound an alarm anyway before being discovered), and "fight back" (attempt to thwart the attacker's activity).

Second, even without total access to the host and without an ability to damage the IDS itself, the intruder can influence the reporting channel. This is because a host-based system must communicate the alerts to the outside parties: email, syslog, SNMP, or other network connection are typically used. Some of these protocols can be disrupted even without having complete access to a host.

Third, host IDS usually presents a bigger administrative and management challenge than network intrusion detection. While it is sufficient to have one NIDS per network segment, HIDS should be installed on every monitored host. That complicates both system configuration and report/alert collection. In addition, having host IDS systems of different classes and from different vendors usually complicates alert correlation and aggregation. Security Information Management solutions can ease the load of analyzing host intrusion detection output.

Fourth, some of the HIDS classes will incur a performance penalty on the protected host. Some vendors report that their kernel-level HIDS incur an extra 5–10% CPU load. Integrity checking is extremely CPU and I/O heavy (if only during the periodic check) due to the underlying cryptographic algorithms.

It should be noted from our honeypot experience, that few of the less competent attackers will bother to check for the presence of the host IDS and/or use any of the above methods to disable it. While typical automated attack kits (autorooter + rootkit) do disable standard UNIX system logging and (sometimes) process accounting, they will not attempt to foil the operation of the integrity checker. One possible explanation is that people who fall victim to such an unsophisticated attack are usually not the users of system integrity checkers. However, in a recent case, the attacker did take steps that accidentally disabled the integrity checking: He simply "rm -rf /"ed the whole machine.

Attacks Against Integrity Checkers

Now let's turn to integrity checkers (IC) and outline some attacks against them together with countermeasures. As was outlined above, the typical IC program computes the checksum and collects information about files ("initialize mode"). Then the program will periodically check for changes (using the "check mode"). In addition, the system admin can update the file signature after reconfiguring the system ("update mode"). Depending on the implementation of the IC program, each of those modes can be attacked.

ATTACKS AGAINST "CHECK MODE"

Trojan Binary

If an IC program uses a standard system binary to check the integrity (e.g., RPM's /usr/bin/md5sum), one can replace the binary to report the right checksum for certain files. Using this simple method, the intruder can hide a small number of files from checksumming, but not from other checks (such as location). For the Linux RPM-based system case, replacing the RPM binary will work just as well.

Countermeasures

This attack is only provided as an example, since most IC programs use much more than just a checksum and often implement their own MD5 algorithm.

Kernel-Level Module (Linux, Solaris, *BSD)

An attacker can deploy a malicious loadable kernel module (LKM) to remap the system calls. As a result the open() call to open a certain file for checking will be redirected to another target. Thus the original file moved to a different location by an attacker will be checksummed. Or, one can leave the open() call to open the original file left in place, but instead redirect the execve() to run the malicious program stored elsewhere (the approach used by twhack.c sample code in the "Bypassing Integrity Checking" article in *Phrack* #51). In addition, remapping some system library (libc) calls can accomplish the same task. The malicious system libraries for Linux and Solaris were observed in the recent system breaches.

Countermeasures

Implementing more checks will require the attacker to remap more and more system calls (which is a non-trivial programming challenge).

Example: Adore v.0.42 vs. AIDE and Tripwire

Adore LKM is a kernel-level backdoor for Linux and FreeBSD, featuring file, process, and connection hiding. Adore remaps fork(), write(), open(), stat() (=get file information), close(), clone() (=like fork()), kill(), mkdir(), and getdents() (=get directory entries) system calls. By default, if you know the filename and location of a file, you will be able to look at it, but it will not show in the directory.

AIDE uses open() to query the files as shown in the above call trace excerpt (obtained by "ltrace -S -f -r -C -s 1000 -o aide-trace aide –check"):

```
22015     0.000249 SYS_open("/etc/security", 67584, 027777753350) = 5
22015     0.000238 SYS_open("/etc/smrsh", 67584, 027777753350) = 5
22015     0.000241 SYS_open("/etc/locale", 67584, 027777753350) = 5
```

Thus, if Adore is configured to hide the presence of a file, AIDE check will not report on the file. On the other hand, Tripwire will still catch the attacker, because it uses read() in its integrity checking after doing an open() based on its own records (and not the getdents() output which is also remapped):

```
22020     0.000075 SYS_read(3, "\177ELF\001\001\001", 1024) = 1024
22020     0.000075 SYS_read(3, "\177ELF\001\001\001", 1024) = 1024
```

It should be noted that by remapping more calls, even this can probably be circumvented.

Faked Report

The attacker might be able to break the email sending functionality and then manually send a faked "All OK" report, modeled after the original report. Admittedly, this attack relies on the intruder's ability to prevent the communication between integrity checking and the reporting station. However, it might require fewer privileges than the full-blown attack, such as "mail" group privileges vs. those of a "root" user. An even more malicious variant of this attack involves replacing the integrity checker binary with an "OK report generator" program, which sends the report to the sysadmin at specified intervals. This involves having root access, but can present a more permanent solution immune to any of the system changes.

If a different networked channel is utilized, it can also be attacked. Consider that having complete control of the target machine, the attacker might initiate any connection (even encrypted) or respond to any connection from an IDS management console. It should be noted that attacks of this kind have not been observed "in the wild."

Countermeasures

Use of secure channels for reporting will stop this attack. Signed email will be considered secure only if someone actually checks the signatures on the reports. Otherwise, faking "signed" reports is just as easy as faking unsigned reports.

Attack Between Checks

While it sounds trivial, if an attacker manages to complete his activities between periodic checks and then restore the system to its original state, the IC program will not report an intrusion. It is impractical to expect hourly integrity checks for most environments (and that would cause heavy CPU utilization for cryptographic checksumming). However, if the original file is replaced, it will likely not be stored in the same disk location and the crime may be discovered.

Countermeasures

Daemon integrity checkers (such as the somewhat obscure "Samhain," available at *http://www.la-samhna.de/samhain/*) do exist. Samhain is an impressive tool that boasts powerful defenses such as an encrypted and compressed executable binary, cryptography support, steganographically shielded configuration files (can be merged with innocent GIF or JPEG images), a deceptive command line, and its own kernel-level hiding kit. These are in addition to secure reporting or information hiding in case the reporting channel is cut off.

As a side note, while some of the crypto-protocols used for integrity checking (such as MD5) were found to have collisions (i.e., different files having the same MD5 checksum), their impact on the security of real-world systems is minor, since it is likely not the weakest link for the typical IC deployment scenario. Still, better integrity checkers, such as Tripwire, use several different algorithms to eliminate this possibility.

ATTACKS AGAINST "UPDATE MODE"

For simple integrity checking programs, attackers will be able to run the "update mode" after modifying the system. Similarly, the attacker can modify the signature database directly. It will work only if the database is stored on the same system that is being checked (not a good idea).

COUNTERMEASURES

Update mode should require a password, and the database should be encrypted (done by some commercial integrity checkers) and, ideally, stored off-site on read-only media (in-depth defense). That will also help to prevent some insider and physical attacks.

ATTACKS AGAINST "INITIALIZE MODE"

The evident attack is that if the system is already trojaned, the initialize mode will create a set of signatures that establish the "compromised baseline." The deployed trojans might also capture the password used for the database.

COUNTERMEASURES

Run the integrity checker right after installation and before connecting to the untrusted network. After creating the database, copy it and store it away from the protected computer.

Now let's consider kernel-level solutions. We will briefly look at Linux StMichael loadable kernel module (LKM), designed to alert on the presence of malicious kernel modules (such as Adore). The module is also able to perform integrity checks on some of the data structures inside the kernel and to detect tampering with various kernel calls. StMichael also conceals itself (using the same tricks as adversaries such as Adore or knark).

For example, if StMichael is loaded and some other kernel module attempts to load in hidden mode and remap system calls, the malicious module will be revealed and (optionally) the changes to a kernel call table will be reversed with the log message:

```
Apr 24 18:32:13 anton kernel: 0(STMICHAEL)
    :Kernel Structures Modified. Attempting to Restore.
```

In case the rootkit was loaded before StMichael, the module might be able to perform detection as well:

```
Apr 24 18:28:59 anton kernel: (STMICHAEL)
    Possible LKM Rootkit Detected during Load.
```

Cryptographically signed system logs (while they sound attractive) do not measure up to a tried-and-true remote logging.

Log Deletion/Modification

Now we are ready to briefly discuss log analyzers. Log analyzers are relatively easy to foil. Most attackers disable system logging and/or wipe system logs and process accounting records. Once root access is achieved, deleting or modifying system logs is easy. Numerous tools (e.g., clean, from THC toolkit, and others) exist to cleanly delete, log, and audit records from text and binary logs. Modern Linux/Solaris rootkits include such tools, and they are automatically activated upon rootkit installation, erasing all traces of log evidence. Now, if remote logging is enabled, the typical rootkit might alert the owner about the fact, but there is nothing it can do about it if the incriminating log messages were already shot across the network over UDP.

COUNTERMEASURES

Remote logging is the most commonly used and reliable measure against log tampering. Using a secure log server or a serial connection to drop off the logs is easy to implement and adds a lot to security.

Cryptographically signed system logs (while they sound attractive) do not measure up to a tried-and-true remote logging. However, it is well known that standard UNIX log transport uses the unreliable UDP protocol. Tools exist to flood the log server with messages and cause it to overflow, crash, or stop receiving messages. In addition, faked data injection can present another risk for some environments. Log processing and correlation engines can be made to reach the wrong conclusion if their correlation logic is known to the attacker. For example, sending faked messages seemingly from the FTP daemon might lead the log analysis program to believe that an FTP session has ended, while in fact the message was crafted by the attacker.

Moreover, a bug in a log monitoring program can lead to a compromise. A recent critical bug in LogWatch enabled attackers to gain local "root" privileges simply by crafting a simple shell script, which abuses temporary files created by LogWatch. More details on the exploit are available at *http://www.securiteam.com/exploits/5OP0S2A6KI.html*.

The obvious conclusion is that to prevent this and other vulnerabilities, log analysis IDS should be run on highly secured log aggregation servers and not on all production machines.

Conclusion

UNIX host security, while somewhat vaguely defined, contributes a lot to maintaining a secure computing environment. This paper introduced some of the issues that should be considered before the deployment of host-based intrusion detection. Let us also note that effective centralized reporting and audit trail analysis will significantly increase the value of host-based intrusion detection. It is crucial to have a central point for the HIDS to report to.

Another promising aspect of host security is application security. Ideally, host IDS should be able to understand the audit trails produced not only by system resources, but also by applications. In this case, a more comprehensive picture of security can be achieved.

not on my watch!

Filtering Options Revisited

Over the past several years, we have all seen "Unsolicited Commercial Email," a.k.a. "spam," grow from an annoyance primarily propagated through Netnews to something that routinely lands in everyone's mailbox. The evolutionary path followed by anti-spam measures somewhat resembles that of network security. Remember when firewalls were (allegedly) optional?

by Strata R. Chalup

President, VirtualNet. Starting as a Unisys 68K admin in 1983, Strata Chalup is now an IT project manager but allegedly has retained human qualities. Her mixed home network (Linux, Solaris, Windows) provides endless opportunities to stay current with hands-on tech.

strata@virtual.net

Attempts to stem the rising tide of spam have had humorous consequences, at least to observers, if not participants. Some articles from our friends across the water point out the ruckus caused by unintended consequences of anti-spam software, ranging from stifling discussions on certain bills in the UK Parliament to rejecting internationalized messages as "inappropriate content." I realize that Welsh isn't for everyone, but, really, that's a bit extreme.

The first-ever Spam Conference recently concluded at MIT and brought together both cutting-edge research and authors of popular freeware and commercial packages. Much good work continues to come out of the conference, and I look forward to seeing the next set of results. Of course, the "Spam Conference" was really about anti-spam methods and the problem of spam, but that's just the way conferences are named.[1]

"Yes, the danger must be growing,
For the rowers keep on rowing,
And they're certainly not showing
Any signs that they are slowing!"
— Willy Wonka

"Analysts believe inbound spam email for the corporation is at least 30% now and will grow to 50% in the next two years."
— Gartner, 2002

"Not on *my* watch!" — everyone to whom I've quoted the above

Tutorials like those provided by ServerWatch can compare and contrast commercial systems, but we've chosen to focus largely on the freeware systems here. Obviously, the best protection is not to get yourself on the lists in the first place, but, as Arlo Guthrie said, "This is not a song about Alice." It is worth mentioning that the spam/anti-spam arms race shows no signs of slowing down. Techniques such as obfuscation with hedge characters or HTML symbol encoding now offer spam-harvesting webbots without the slightest hiccup. Embedding mailto links or email addresses in a protective bezoar of JavaScript is good protection now but probably the next bit of digestive evolution for the e-bots. Like server-side generation of text images, this is also an accessibility issue, foiling conventional text-to-speech systems as well as address-harvesting 'bots.

As always, things will get worse before they get better. A recent MessageLabs report shows that as filtering options improve, spammers (and virus writers!) are increasingly targeting loopholes in our mail clients and mail-handling procedures. For example, what if you get an attachment called *our-new-house.jpg.exe.jpg*?

To quote from the report, "The malware relies on especially crafted email headers, creating an attachment with three file-extensions. . . . The first extension . . . is visible to

1. I still haven't seen a more unfortunate name than that posted on a call for papers at the MIT Psychology Department, where I once managed systems: "(n)th Annual International Invitational Traumatic Head Injury Conference." Ouch!

The issues of anti-spam and anti-virus are increasingly converging.

the email user, and is intended to persuade them that the attachment is "safe." The final extension . . . is used by Outlook Express to set the icon to represent the application for opening the attachment. . . . However, the unusual middle extension (.EXE) is used by Outlook Express to determine how to launch the attachment; therefore an .EXE file will be executed if a user double clicks on an infected attachment." The next generation of spam harvesting tools will probably include viruses which gather spam directly from people's address books, so the issues of anti-spam and anti-virus are increasingly converging.

The more things change, the more they stay the same. Email anti-spam technology is recapitulating the ontogeny of Usenet anti-spam technology. Aren't we overdue for the Breidbart Index Filtering on ISP mail gateways and email security products? (*http://www.stopspam.org/usenet/mmf/breidbart.html*). Blacklists have been around forever, and whitelists are gaining in popularity. Two years ago there were only one or two freely available quasi-automated whitelisting systems, while now a double handful can be found, and even a company or two staking its future on a special type of whitelisting. Sophisticated pattern matching is being augmented by even more sophisticated heuristic-based Bayesian modeling. Service providers are even attempting to require authentication and/or control of accessible servers to try to stop spam at its source. Let's take a look!

Follow the White(list) Rabbit

One article mentioned that a favorite trick of randomizing spammers is to twiddle with the comments in HTML-formatted spam. The message looks identical to the unlucky recipient, but generates a different checksum. The author's response was that "No one I care to talk to sends mail as HTML" and that his practice is to "direct HTML mail to my spambox."

We should all be so lucky! The reality is that shunting HTML-formatted mail to a spam box only works tolerably if accompanied by aggressive whitelisting of friends, family, and coworkers. The primary disadvantage of whitelisting, of course, is the onus on you, the recipient, to keep the whitelists updated as people change their addresses, send mail from other accounts while traveling, and the like. Fortunately, there are a plethora of options from which to choose, many of which are listed in this article's listing of links.

Taking the concept of whitelisting to perhaps its most extreme level is the Habeas system. This unusual system rests on modern patent and trademark law and will be truly useful only when large numbers of persons start using it. As you might thus expect, it is currently free to individuals and service providers. Commercial entities must pay a licensing fee but, more importantly, jump through some well-defined hoops. Habeas has copyrighted a specific haiku, and it has a patent pending for their use of "protected" text, called a Warrant Mark, in message headers to provide authentication. It is unclear from their Web site if the patent includes their specific blacklist of noncompliant entities.

To be a Habeas-compliant entity, one must only send messages containing the special text headers to recipients who have truly opted in to receiving the message. Spammers who use the Warrant Mark in their mails are liable for prosecution under good old-fashioned copyright and patent law. Habeas claims to have created a structure in which a traditional legal framework is sufficient for prosecution, with no reliance on newfangled and often confusing cyberlaws. If widely adopted, the system would provide a combination of guaranteed marking of non-spam mail and a way to go after spammers who abuse the Warrant Mark.

Why is this supposedly better than generic whitelisting? The company's FAQ reminds people that whitelists cannot detect spammers forging popular "From" addresses, such as notification addresses from retailers. To be scrupulously fair, a forged set of headers containing the Habeas Warrant Mark would also not be detected, unless sent by a repeat offender already blacklisted. However, you may feel better about viewing it, given that your complaint (to Habeas) will actually cause something to happen, namely blacklisting and an aggressive legal pursuit of the spammers for infringement.

Down a Different Rabbit Hole: RFC 2476

Service providers, and an increasing number of corporations, are requiring authentication to internal mail servers and blocking access to port 25 of external servers. Together these steps can certainly reduce the amount of spam generated at a typical huge ISP, but they can also really cramp your style if you are traveling and would like to preserve your email independence. "Hmm," you say, "sounds like it's time to find another port." Exactly so, but as an Upstanding Net Citizen you worry about sending mail to *Adam.West@WayneManor.org* through a port other than the well-known service port for SMTP. Holy protocol, Batman! Enter RFC 2476 to the rescue!

The issue is not really one of which port to use, although the RFC 2476 does define port 587 as the WKS port for message submission. The primary focus of the RFC is to distinguish between *message transport*, in which an MTA must not meddle with certain aspects of the message, and *message submission*, where it may be useful or needful to alter or add to a message. The first two reasons given in the RFC are extremely germane to this discussion, namely:

- Implement security policies and guard against unauthorized mail relaying or injection of unsolicited bulk mail
- Implement authenticated submission, including off-site submission by authorized users such as travelers

In his excellent series of articles, "RFCs for the Rest of Us," Paul Boutin discusses RFC 2476 in detail, along with RFC 2554 (SMTP Authentication) and RFC 2505 (Anti-Spam Recommendations).

Communities and Checksums

Vipul's Razor (v2) is a checksum-based method of tagging messages as potential spam. Netnews administrators may recognize this methodology from various NNTP filtering systems. Razor has the familiar advantages of digest or checksum-based approaches over pattern-matching rule-based systems, most notably lower computational overhead and small data sets. Of course, there is a glaring disadvantage – that randomizing a small part of the message body will change the checksum and let spam sneak in the door.

A complex mesh network of hosts is aggregated under a DNS zone used by Razor Agents to find Razor Discovery Servers. The Razor Agents query Razor Discovery Servers to find the Razor Catalogue Servers (for razor-check(1)) and Razor Nomination Servers. The default is *razor.cloudmark.com*, but the appearance of a GPL'd version of Razor called "Pyzor" and a separate initiative called the Distributed Checksum Clearinghouse (DCC) now gives ET somewhere else to phone home. In practice, SpamNet probably represents the largest user community, and it is reporting solely to Cloudmark. Spam Assassin would clearly be next in line, and while it offers reporting checksums to all three services, it's not clear how widely this has been adopted. For the curious, there is a description of the Razor reporting protocol at *http://www.stearns.org/razor-caching-proxy/razor2-protocol*.

Cloudmark's SpamNet is one of those "good news, bad news, good news" deals. The good news is that it's free. The bad news, for many of us, is that the only currently supported client is Microsoft Outlook 2000/XP/2002. But the good news beyond that is that SpamNet is essentially the pseudo-commercial arm of Vipul's Razor, as Vipul is one of Cloudmark's founders. Cloudmark is the primary aggregator of Razor/SpamNet data, and it's worth mentioning that a for-pay service, Cloudmark's Authority, leverages the data gathered by the free SpamNet community service.

Cloudmark has taken Razor's data-gathering one step further – using Bayesian classification, they turn the data into the somewhat loftily named "spamGenes" and "spamDNA." Their claim is that there are only 150 spamGenes and that their method consumes vastly fewer resources at the gateway. Let's see, there's "free," "v*g*r*a," "mrs mobuto sese seko," and, um, 147 more. One clue emerges from a *Wall Street Journal* article on Authority – namely, that the software concentrates on "the marketing message . . . it's how they make money and it doesn't change a lot." Neither does Authority; updates are made available every 30 to 60 days "in the form of spamDNA cartridges." Do those count as biohazards? Only if you're a spammer, I guess.

As does its predecessor, Razor v2, the SpamNet client preserves individual user privacy by generating a "fingerprint" or digest of a spam message and sharing only the fingerprints among SpamNet users. However, Cloudmark's Web site mentions the existence of a "Truth Evaluation System (TES)," which apparently rates each SpamNet user according to various factors, including volume, relevance, and accuracy. To quote the site, "Simply, long-time, trusted-user reports carry more weight in spam identification than new, untested reports. When a SpamNet member makes a good report, their trust rating is increased." If a user realizes that a spam that he or she marked as "Block" is actually a legitimate email, the user may "Unblock" it and get back their good SpamNet karma. This is the same mechanism employed by Razor.

"Heterodyne Portable Claw: Use Only for Good."

I have to put on my "virtual Peter Neumann hat" here and talk about some of the risks of systems like SpamNet. A virtuous privacy policy is no guarantee that one's data will not be used for marketing. It is only a guarantee that the current corporate structure will not use that data. I hope that Cloudmark has some stringent policies in place about whether their TES is a "corporate asset" or not. Currently, Cloudmark appears to be privately held, but I doubt that is their long-term strategy. If a less enlightened corporate entity were to obtain control of Cloudmark's assets through an acquisition, it would be very easy for them to build a truly impressive marketing database.

They could use existing ways of mining personal data from Web sites in the context of offering a SpamNet update or, perhaps, in the course of collecting the normal data from a SpamNet client. They might be able to cross-correlate with an existing marketing database such as DoubleClick or MSN. Since a reputation system is employed in the TES, each SpamNet client must have a unique identifier. User-reported spam fingerprints could be correlated with full-text spam, which in turn could be demographically sorted as targets via conventional marketing analysis. One could certainly create an interesting reverse-engineered demographic database with Cloudmark's TES and a sufficiently large sample space of spam, such as the Ciphertrust spam archive.

Sound far-fetched? Companies such as DoubleClick make their living doing very similar analysis, based on Web and email cookies. I am not saying that one should not use SpamNet. I am saying that, in the spirit of RISKS-Digest, one should understand what the technology enables. When this scenario was described to a highly-placed source within Cloudmark, the danger was discounted as implausible. Then again, who believed five years ago that someday all your old radical college Usenet postings would be searchable, or that websites which you've never surfed before would greet you by name based on shared marketing profiles?

Bayesing at the Moon

You can't pick up an IT press article about anti-spam systems these days without encountering the buzzwords "Bayesian," "heuristics engine," or something similar. Hey, these were all around years ago on Usenet. So just how did we "forget" Bayesian filtering for so long? Paul Graham provides an excellent summary in his report to the MIT Spam Conference. Lack of acceptance of a "miss rate" of 92% with 1.16% false positives seems to have been the key factor. What Paul and others found is that the direct application of the Usenet technique ignored the message headers, which can arguably be said to be less meaningful in the NNTP context than in that of SMTP. When headers were factored in, the miss rate dropped to 99.5% with less than 0.03% false positives, applying the identical techniques previously used by Pantel and Lin.

Two secondary factors were the adaptive, or learning, capability of the filters, and the use of weighted tokens. The accuracy improves noticeably when the sample size is increased. Getting the accuracy rate that high involved putting a great deal more spam through the system, yielding impressive results. Additionally, by choosing the top 15 or so tokens to weight most heavily, the system can better deal with spams that, as Graham puts it, "tell you their life story" in the course of getting to the punch line.

A radically different approach to Bayesian filtering involving regular expression matching rather than tokenized input, the CRM114 system by Bill Yerazunis shows that we haven't even begun to run out of fire power to throw at the problem. The Controllable Regex Mutilator, to quote its home page, "offers sparse binary polynomial matching with a Bayesian Chain Rule. . . . Accuracy of the SBPH/BCR classifier has been seen in excess of 99 per cent, for 1/4 megabyte of learning text. In other words, CRM114 learns, and it learns fast." Yow!

The Swiss Army Knife Approach

The interestingly disjoint lists of server-side anti-spam tools at various Web sites suggest either an uninformed or highly opinionated general admin populace, or a highly insoluble problem that fits everybody like a bad pair of shoes. Are anti-spam software developers treading out their own Shoe Event Horizon?

One thing that many of these tools have in common is that they deploy mail through procmail, and then let loose with a whole arsenal of techniques. Filtering FAQs abound, but we won't reinvent the wheel, we'll just cite it in the references. Lately, even procmail substitutes are cropping up – if you're tired of procmail, try a substitute handler such as Salmon, which wraps some basic setup tasks along with the procmail functionality and an anti-spam engine.

LINKS

MPs discussions censored by protective filters:
http://www.theregister.co.uk/content/6/29175.html

Filter woes continue for MPs:
http://www.theregister.co.uk/content/6/29199.html

Triple (extension) threat:
http://www.messagelabs.com/viruseye/report.asp?id=130

ServerWatch Tutorial and Product Comparison:
http://www.serverwatch.com/tutorials/article.php/10825_1567361_2

O'Reilly article on spam harvest prevention:
http://www.macdevcenter.com/pub/a/mac/2002/11/01/spam.html

RFC 2476: Message Submission:
http://www.faqs.org/rfc/rfc2476.txt

Paul Boutin's "RFCs for the Rest of Us":
http://www.sendmail.net/rfcintro.shtml

maildrop:
http://www.flounder.net/~mrsam/maildrop/

Habeas "Sender Warranted Email":
http://www.habeas.com/faq/index.htm

Active Spam Killer (whitelist):
http://paganini.net/ask/

Tagged Message Delivery Agent (whitelist):
http://tmda.net/

Mail DeSpammer (reactive whitelist):
http://www.laas.fr/~felix/despam.html

The Infamous Big Brother Database
http://bbdb.sourceforge.net/
http://www.jwz.org/bbdb/

Filtering FAQ Fun for All and Sundry:
http://mip.ups-tlse.fr/~grundman/procmail/faq.html

Salmon (procmail++ ? YMMV):
http://is.rice.edu/~wymanm/smn/index.html

Spambouncer:
http://www.spambouncer.org/

2003 Spam Conference at MIT:
http://spamconference.org/

Bill Yerazunis' CRM114 system:
http://crm114.sourceforge.net/

Seriously technical goodies on filtering here:
http://www.paulgraham.com/bayeslinks.html

How Bayesian filtering evolved past Usenet:
http://www.paulgraham.com/better.html

Scads of clients for indiv & server ops:
http://email.about.com/cs/bayesianspamsw/

The Shoe Event Horizon
http://www.csua.berkeley.edu/~dxu/econ/shoe.html

But wait, there's more . . . :
http://dmoz.org/Computers/Software/Internet/Servers/Mail/AntiSpam/

junkfilter:
http://junkfilter.zer0.org/

Usenet anti-spam resources – all your old buddies like CleanFeed and SpamHippo and the like:
http://www.exit109.com/~jeremy/news/antispam.html

Cloudmark's SpamNet client (free):
http://www.cloudmark.com/products/spamnet/learnmore/spamnet.php

Cloudmark's Authority server software ($):
http://www.cloudmark.com/products/authority/

Web log article on Cloudmark, with screen shots:
http://www.emergic.org/archives/2003/01/10/

Girl Genius – Go Agatha!:
http://www.studiofoglio.com/girlgenius.html

Vipul's Razor:
http://razor.sourceforge.net/

Spam Assassin:
http://spamassassin.org/
http://spamassassin.taint.org/

The Outer Limits:
http://www.innermind.com/outerlimits/info/olopen.htm

Pyzor (Python, GPL version of Razor):
http://pyzor.sourceforge.net/

Distributed Checksum Clearinghouse:
http://www.rhyolite.com/anti-spam/dcc/

Email Sanitizer and Esd-l:
http://www.impsec.org/email-tools/procmail-security.html
http://www.spconnect.com/mailman/listinfo/esd-l

Brian Hatch's "Filtering Email with Postfix and Procmail" series (includes code examples). Parts 1 & 2 are Postfix-specific; 3 & 4 cover procmail and integration with various packages like Razor and Spam Assassin:
http://online.securityfocus.com/infocus/1593
http://online.securityfocus.com/infocus/1598
http://online.securityfocus.com/infocus/1606
http://online.securityfocus.com/infocus/1611

Email on SOHO Networks:
http://www.unixreview.com/documents/s=7460/uni1032893910897/ur0209o.htm

Ricochet Spam Handler:
http://vipul.net/ricochet/

A good example of this kind of technology is John Hardin's Email Sanitizer, as featured on the Email Security Discussion list (Esd-l). Introduced in 1999, it's a quiet example of a mature, refined, and ultra-configurable procmail rule suite that lets you pick the best of the best and apply it. Esd-l is also a good place to pick up breaking news about new attacks, such as the triple-threat extension trick mentioned at the beginning of this article.

One of the most popular tools, and one increasingly shipping quietly under the hood of many commercial anti-spam software suites and appliances, is Spam Assassin. It is truly the adaptive kitchen sink or Swiss Army knife of the cumulative filtering tools. Spam Assassin filters spam using a combination of traditional methods, including header and body checks, blacklists, and whitelists. On top of these metrics, it uses Vipul's Razor to score messages. Individual tests are weighted, as is the threshold at which the system decides "OK, this is spam."

Unlike the old *Outer Limits* TV show, you control the horizontal, you control the vertical, since weighting and threshold are user-adjustable. For instance, Spam Assassin now comes with a weighting for the Habeas Warranted Email service mentioned earlier; defaults are set to award a Habeas-compliant message a more beneficial status. Meanwhile, the Bayes system provided by the "sa-learn" facility keeps trying to predict what you consider spam vs. messages you want to see.

Spam Assassin's fans claim over 99% accuracy, but many first-time users report very different results. The key seems to be aggressive whitelisting, especially of mailing lists to which you have voluntarily subscribed. An unfortunate gap in the coverage results, since one source of spam for many of us is non-technical hobby or interest lists which may be spammed to reach subscribers. It might be worth experimenting with recursive calling of differently configured Spam Assassin instantiations, or combining Spam Assassin with some other program that will then sift your less well-behaved lists for secondhand spam.

But Wait, There's More!

A dizzying array of spam-prevention technologies exists to combat spam at the server level, for your home, office, or whole organization. Rather than attempt an exhaustive survey, I've included links to some of the more interesting ones. To save your cut-and-paste macros some work, we'll post the links on VirtualNet, so you can just bookmark-and-go. *http://www.virtual.net/Ref/resources.html* contains a bibliography of all my articles, and will be updated with this one.

One final note: To round out his contribution to spam fighting, the talented Vipul also wrote a spam tracer and handler called Ricochet to deal with spam that successfully runs the formidable gauntlet we've set up here. Enjoy!

compressing Web output using mod_gzip and Apache

Web-page compression is not a new technology but has only recently gained higher recognition in the minds of IT administrators and managers because of the rapid return on investment it generates. Compression extensions exist for most of the major Web server platforms, but in this article I will focus on the Apache and mod_gzip solution.

by Stephen Pierzchala

Stephen Pierzchala is senior diagnostic analyst for Keynote Systems in San Mateo, CA. He spends his time reminding Web developers about the need for Web performance.

stephen@pierzchala.com

The idea behind GZIP-encoding documents is very straightforward. Take a file that is to be transmitted to a Web client and send a compressed version of the data rather than the raw file as it exists on the file system. Depending on the size of the file, the compressed version can run anywhere from 20% to 50% of the original file size.

In Apache, this can be achieved using a couple of different methods. Content negotiation, which requires that two separate sets of HTML files be generated – one for clients that can handle GZIP-encoding and one for those that can't – is one method. The problem with this solution should be readily apparent: There is no provision in this methodology for GZIP-encoding dynamically generated pages.

The more graceful solution for administrators who want to add GZIP-encoding to Apache is the use of mod_gzip. I consider it one of the overlooked gems for designing a high-performance Web server. Using this module, configured file types – based on file extension or MIME type – will be compressed using GZIP-encoding after they have been processed by all of Apache's other modules, and before they are sent to the client. The compressed data that is generated reduces the number of bytes transferred to the client, without any loss in the structure or content of the original, uncompressed document.

mod_gzip can be compiled into Apache as either a static or dynamic module; I have chosen to compile it as a dynamic module in my own server.[1] The advantage of using mod_gzip is that this method does not require anything to be done on the client side to make it work. All current browsers – e.g., Mozilla, Opera, Internet Explorer – understand and process GZIP-encoded text content.

1. The servers used for this article were from the Apache 1.3.x family.

On the server side, all the server or site administrator has to do is compile the module, edit the appropriate configuration directives that were added to the httpd.conf file, enable the module in the httpd.conf file, and restart the server. In less than 10 minutes, you can be serving static and dynamic content using GZIP-encoding without the need to maintain multiple code bases for clients that can or cannot accept GZIP-encoded documents.

When a request is received from a client, Apache determines if mod_gzip should be invoked by noting if the Accept-Encoding: gzip HTTP request header has been sent by the client. If the client sends the header, mod_gzip will automatically compress the output of all configured file types when sending them to the client.

This client header announces to Apache that the client will understand files that have been GZIP-encoded. mod_gzip then processes the outgoing content and includes the following server response headers:

```
Content-Type: text/html
Content-Encoding: gzip
```

These server response headers announce that the content returned from the server is GZIP-encoded, but that when the content is expanded by the client application, it should be treated as a standard HTML file. Not only is this successful for static HTML files, but it can be applied to pages that contain dynamic elements, such as those produced by Server Side Includes (SSI), PHP,[2] and other dynamic page-generation methods. You can also use it to compress your Cascading Style Sheets (CSS) and plaintext files. My httpd.conf file sets the following configuration for the file types handled by mod_gzip:

2. PHP can also be compressed using the integration with the native ZLIB compression libraries. This integration can be built in at compile time and activated through the php.ini file.

```
mod_gzip_item_exclude       file        \.js$
mod_gzip_item_exclude       mime        ^application/.*$
mod_gzip_item_exclude       mime        ^image/.*$
mod_gzip_item_include       file        \.html$
mod_gzip_item_include       file        \.shtml$
mod_gzip_item_include       file        \.php$
mod_gzip_item_include       file        \.txt$
mod_gzip_item_include       mime        ^text/.*$
```

I have had limited success compressing other file formats, mainly because Microsoft's Internet Explorer appears to examine the "Content-Type" header message before it examines the "Content-Encoding" header message. So, say you configure your server to GZIP-encode PDF files using the following mod_gzip directives:

```
mod_gzip_item_include       file        \.pdf$
mod_gzip_item_include       mime        ^application/pdf$
```

When downloaded by Mozilla and Opera, the PDF files are immediately decoded and passed to the appropriate helper application. These browsers know to decode all GZIP-encoded content before passing it along to the appropriate helper application.

However, Internet Explorer simply passes the GZIP-encoded content directly to the PDF reader without first decoding it. A quick rummage through newsgroup archives turned up evidence that this "feature" has been in Internet Explorer since at least 1997. I chalk it up to the lingering integration of browser and operating system through the Component Object Model (COM). This has a potentially detrimental impact on the Web community as a whole range of additional file types could be compressed if this bug was fixed.

How beneficial is sending GZIP-encoded content? In some simple tests I ran on my Web server using WGET, GZIP-encoded documents showed that even on a small Web server there is the potential to produce a substantial savings in bandwidth usage. For *http://www.pierzchala.com/bio.html*, uncompressed file size was 3122 bytes, compressed was 1578 bytes. And for *http://www.pierzchala.com/compress/homepage2.html*, uncompressed file size was 56279 bytes, compressed was 16286 bytes.

Server administrators may be concerned that mod_gzip will place a heavy burden on their systems as files are compressed on the fly. I argue against that, pointing out that this does not seem to concern the administrators of Slashdot (*http://slashdot.org/*), one of the busiest Web servers on the Internet, who use mod_gzip in their very high-traffic environment.

The mod_gzip project page is located at SourceForge: *http://sourceforge.net/projects/mod-gzip/*.

isn't that a little risky?

by Ray Swartz

Ray Swartz ran his own computer training and consulting company for 20 years, until stepping away from gainful employment in 2000. Watching the stock market and his retirement portfolio move up and down like a yo-yo has given him a new appreciation of his tolerance for risk.

raybo@idiom.com

In previous columns, I wrote that the best way to ensure that you retire in style is to create a financial plan that enables it. This means calculating your current income, taxes, expenses, and investments and estimating what these might be for the future you want to live. Since very few people can fund their retirement out of existing savings, almost all financial plans will rely on investments to grow over time to cover future requirements.

No matter what you do with the money you are saving for retirement, you are taking some risk. The risk may be due to corporate malfeasance, market fluctuation, inflation, interest rates, or world events. Whatever the risks, the key to good investing is to minimize the risks you take to get the return you need.

Isn't Investing Just Like Gambling?

Given these risks, it is common to hear investing compared to gambling. While there are some striking similarities, such as making money and losing one's "investment," there are two important differences.

First, a gambling bet either pays off a specified amount or you forfeit your entire stake. If you win, you end up with a known amount more than you started with. If you lose, you're left with nothing.

An investment, on the other hand, is seldom an all-or-nothing affair. It is rare when an investment results in a total loss. Also, you don't know at the beginning how much you might get if you "win." What's more, there are lots of intermediate outcomes and you can cash in an investment, winner or loser, whenever you choose.

In addition, many stocks and bonds make periodic payments in the form of interest or dividends. Some investments are made primarily for these payouts. Dividends and interest also help cushion any downward movement in the value of these investments.

Second, the underlying probabilities of a specific bet can be calculated precisely. Because these probabilities identify *long-term* outcomes, a gambler relies on irregularities in the short-term distribution to achieve a "profit."

Investment prices, however, are based on market forces that can be estimated but not measured precisely. An investor relies on the belief that in the future the demand for a specific investment will outstrip its supply and the price will rise, be it a share of stock, the interest paid by a bond, or a piece of real estate. Unlike a dice game, where a die always has six sides, the marketplace is constantly undergoing change. It is not possible to say with any certainty what the price of an investment will be in the next few minutes, let alone months or years.

Quantifying Risk

The probabilities associated with two six-sided dice are known, so a gambler is aware of the risks being taken when betting at a crap table. However, the risks of buying 100 shares of a company's stock or a $1,000 bond are not quite so obvious. Since riskier investments, like chancier bets, carry a higher possibility of loss, it makes sense that they also should compensate by providing larger returns. But, without a way to quantify risk, it is impossible to know if an investment's risk justifies its return.

A great deal of theoretical work and data collection have been undertaken to define investment risk. For the most part, this effort has focused on defining equations that

Simply knowing how much an investment's value varies is of limited use. More useful is comparing an investment's volatility with that of the overall market.

statistically fit the data collected and assigning Greek letters to various coefficients in the results.

Since statistics are involved, some assumptions get made. This research assumes that the past accurately predicts the future and that variations in investment values are normally distributed. Data seem to support these.

In the end, two values have emerged to represent an investment. One is the investment's expected return, which is calculated as its average gain/loss over time. The second measurement is the investment's volatility: that is, how much its value varies, calculated by taking the standard deviation of the investment's gains/losses over time.

Simply knowing how much an investment's value varies is of limited use. More useful is comparing an investment's volatility with that of the overall market. This ratio is represented by the Greek letter beta (β). A β of 1.0 means that a stock investment is as volatile as an underlying index (a 5% move in the index would mean a 5% move in this stock). A β of 2.0 means the stock's price moves twice as much as the index. An investment with a β of 0.5 would move half as much as the market. A company's β is considered a measure of its risk.

Two stocks offering the same expected return at the same β could be considered equivalent investments (though other factors would go into such a determination). Investments might offer the same expected return but have different betas, or the same βs but different returns. A higher β means that an investment's value will move around more than one with a lower β.

Suppose your financial plan calls for a return of 8.25% on your retirement assets. Knowing the return you need allows you to select how much risk you want to take in order to get it. Let's say that the market's 50-year average return is 9.5% (it isn't). Since you only need an 8.25% return, you could seek investments with βs of less than 1.0 and have a portfolio with less volatility than the overall market.

You might also devise a strategy that combines investments of different risk levels to further reduce the overall risk of your portfolio. For example, you might combine some government bonds with a return of, say, 6.0% with a market index fund to achieve your 8.25% return. βs for individual stocks and mutual funds are readily available from investment counselors (another source is *http://www.morningstar.com*).

Different Kinds of Risk

Total investment risk is a combination of different risks all acting together to create volatility. There are many reasons why investment values go up and down: A company is going through good or bad times (company risk); a company's industry is having a hard time, and all companies in that industry are affected (industry risk); the entire stock market is moving up or down and stocks are doing the same regardless of their individual merits (market risk).

A similar kind of analysis applies to bonds. Bonds are a loan of money to a company or government in exchange for a preset rate of interest for a specified period of time. Bonds are bought and sold on the open market just like shares of stock, except they don't confer ownership in the underlying company (or government :-). Since bondholders have no ownership stake in the company, any factor that affects the payment of interest will move the price of the company's bonded debt. To help guide buyers of bonds, rating agencies such as Standard & Poor's and Moody's have devised scoring systems that evaluate a company's ability to meet its bond obligations (default risk). These ratings are equivalent to a stock's β, and a bond's value is greatly affected by the company's debt rating.

While the market value of a bond varies, there is no volatility to the return of a bond if you hold it to maturity. As long as the company doesn't default, you will continue to get the same interest for the life of the bond. However, if you need to sell a bond before it is due, you have to find a buyer in the open market.

What a buyer is willing to pay for a bond depends on several factors, such as the interest the bond pays, the company's bond rating, and the overall sentiment about interest rates. In addition to these factors, the market price of a bond is greatly affected by the current rate of interest for this type (length and quality) of debt.

Let's suppose you own a $1,000 bond paying 5% interest. If the going rate for bonds like yours is 6%, then a buyer will discount the value of your bond until it, too, is paying 6%. The discount is easy to calculate. Your bond pays $50 per year and $50 is 6% of $833.33, which is what a buyer would be willing to pay for your bond. Because interest rates rose (interest rate risk), you've lost $167.67 (assuming you bought the bond for $1,000). Incidentally, this is why the value of a bond moves inversely to the interest rate. If the interest rate falls, a buyer will pay a premium for your bond, instead of demanding a discount!

Another problem that bondholders have to consider is inflation. As inflation rises (inflation risk), the interest a bond pays buys less. Also, the return of the principal will be in inflated dollars. Inflation risk is not such a problem with stocks, as companies can raise prices to offset the impact of higher costs.

It might seem like a no-brainer to fund one's retirement by buying bonds that yield the interest rate required by your financial plan. But there is a catch. Your financial plan assumes that you earn the specified return continuously. Thus, when you receive a bond's interest, you have to reinvest that interest in accordance with your plan. However, interest rates move every day (interest rate risk, again), and you might not be able to reinvest that interest at the necessary percentage rate without taking on more risk. An example may make this clearer. Your $1,000 bond paying 5% gets you $50 a year in interest. Today, getting 5% interest requires either very long-term or high-risk bonds. Interest rate and inflation risks make it hard for a portfolio 100% invested in bonds to maintain a high rate of return over the long term.

Not all risks are the same. Just because two investments have the same amount of volatility doesn't mean they have the same pattern of volatility.

Getting the Most for Your Risk

As I described above, not all risks are the same. Just because two investments have the same amount of volatility doesn't mean they have the same pattern of volatility.

Consider two stocks that are in the same industry – say, GM and Ford. Since they are both in the same business, their stocks will likely move together. When one is up or down, so is the other. If there is a slump in car buying (industry risk), both stocks will go down. Note that due to company risk, these two stocks will not move in lockstep.

Now, consider stocks in different businesses, say GM and Levi Strauss. Since these stocks are not in the same industry, there will be times when one is down but the other is up. The end result for us is less overall volatility!

If Ford, GM, and Levi Strauss all have the same expected return and βs, a portfolio containing both GM and Levi Strauss will have the same return but less overall volatility than a similar one containing Ford and GM or 100% of just one of the companies. By buying two stocks instead of one, we lower company risk. By having stocks in different industries, we also lower industry risk.

By picking our investments wisely, we can reduce the overall risk of our retirement portfolios without having to accept lower returns! In investment lingo, this is called diversification. It is worth noting that the downside of diversification is that the combined portfolios have less range, both high-end and low-end, than undiversified investments. That is, the total upside and downside potential is less when you diversify your risk. If your plan is solid, losing this bit of unlikely upside will have no impact on your future.

How Do I Use This Information?

While all this sounds good, how does it work in the real world? Calculating how two stocks vary means calculating the statistical covariance of their stock movements.[1] This can be done if you have all the data and a good statistical package. However, there are thousands of stocks, and building a complete covariance table for each one would be way too much work. What is more common is for an investment's covariance to be calculated to some standard index, like the S&P500. This information is available from financial advisors and from the Net (again, at *http://www.morningstar.com*, among other sites).

1. Covariance is a statistical calculation that measures how two sets of values vary with one another. For investment purposes, covariance results range from +1 to -1, with +1 for two investments that move together 100% of the time and -1 representing two investments that move exactly opposite to one another.

If you invest in individual stocks, you need to be careful to compare the companies you invest in by industry and covariance. Keep in mind, though, that buying shares in just a few companies still exposes you to significant company risk, since a meaningful percentage of your portfolio may be concentrated in a few stocks. To gain the benefit of diversification, many commentators suggest holding at least 10 different stocks. The most common way to minimize company risk is to invest in mutual funds. Since mutual funds buy into many companies, your exposure to any single company is greatly reduced.

There are other ways you might want to diversify. Common investment advice says to diversify geographically by investing in the companies of different countries. This not only allows you to lessen the impact of holding all your money in a single currency (currency risk) but also takes advantage of covariance between the economies of different global areas. However, international investing has its own unique risks, such as political instability, market manipulation, and outright fraud.

Another way to diversify is by the size of the companies you invest in. Typically, stocks are divided into three classes: large-cap, mid-cap, and small-cap. The word "cap" is short for "capitalization" and refers to the total value of a company, which is calculated by multiplying the number of shares outstanding by the share price. Large-cap companies tend to be established, well-financed industry leaders, whereas small-cap companies are lesser known and often more vulnerable to the vagaries of the marketplace. Stocks tend to be more highly correlated with others in their same class than with stocks in different classes. As such, different classes of stocks don't always move in the same direction, so they offer further diversification opportunities. Mutual funds usually clearly identify what class of stocks they buy, so it is easy to determine the capitalization mix of your portfolio.

Since stocks and bonds tend to move in opposite directions, another way to reduce your portfolio's volatility is to hold both stocks and bonds. In theory, in good times the stocks go up, in bad times the bonds go up (and still pay interest)!

As a counterpoint to all this talk about diversification, it is possible to over-diversify. In this case, your portfolio is spread around in so many different investments that you can't take advantage of a strong move in any one of them. Also, the more investments you have, the more incidental costs you pay, which further decreases your return.

How can you know if your portfolio is taking too much risk for the return it is generating? The best advice is to talk with your investment advisor. Financial planners often have software programs that can dissect your portfolio and compare its expected return to its estimated risk.

Lastly, there is some risk that can't be diversified away: market risk. Some events such as natural disasters, terrorist attacks, war outbreaks, or economic meltdowns move all markets down. However, the effect of such shocks is often short-lived, especially if the underlying economies remain well-ordered and responsive. The only way to avoid market risk is to hold cash or equivalents, such as bank CDs. But these investments have risks, too, such as interest rate fluctuations and inflation.

In the end, no matter what you do with your investment dollars, you take some risks. While risk can't be avoided, it can be minimized.

the bookworm

by Peter H. Salus

Peter H. Salus is a member of the ACM, the Early English Text Society, and the Trollope Society, and is a life member of the American Oriental Society. He is Editorial Director at Matrix.net. He owns neither a dog nor a cat.

peter@matrix.net

BOOKS REVIEWED IN THIS COLUMN

COMPUTER SECURITY: ART AND SCIENCE
MATT BISHOP
Boston: Addison-Wesley, 2003. Pp. xli+1084. ISBN 0-201-44099-7.

FIREWALLS AND INTERNET SECURITY
WILLIAM R. CHESWICK, STEVEN M. BELLOVIN, AND AVIEL D. RUBIN
Boston: Addison-Wesley, 2003. Pp. 430. ISBN 0-201-63466-X.

802.11 SECURITY
BRUCE POTTER AND BOB FLECK
Sebastopol, CA: O'Reilly, 2003. Pp. 176. ISBN 0-596-00290-4.

SHARING EXPERTISE
MARK ACKERMAN ET AL., EDS.
Cambridge, MA: MIT Press. Pp. 438. ISBN 0-262-01195-6.

BGP
ILJITSCH VAN BEIJNUM
Sebastopol, CA: O'Reilly, 2002. Pp. 272. ISBN 0-596-00254-8.

DNS & BIND COOKBOOK
CRICKET LIU
Sebastopol, CA: O'Reilly, 2003. Pp. 222. ISBN 0-596-00410-9.

C POCKET REFERENCE
PETER PRINZ AND ULLA KIRCH-PRINZ
Sebastopol, CA: O'Reilly, 2003. Pp. 134. ISBN 0-596-00436-2.

PHP POCKET REFERENCE, 2D ED.
RASMUS LERDORF
Sebastopol, CA: O'Reilly, 2002. Pp. 132. ISBN 0-596-00402-8.

ESSENTIAL SYSTEM ADMINISTRATION POCKET REFERENCE
AELEEN FRISCH
Sebastopol, CA: O'Reilly, 2003. Pp. 137. ISBN 0-596-00449-4.

WORD POCKET GUIDE
WALTER GLENN
Sebastopol, CA: O'Reilly, 2003. Pp. 143. ISBN 0-596-00445-1.

There are many books to talk about this month, as well as two I'm shoving into my "history" column. The holidays gave me lots of time both to read and to ignore football. In fact, not watching football contributed to my time for reading.

Security

I reported on the security symposium last December. Little did I know that two important volumes would overburden my postman so soon.

The larger of these is Matt Bishop's monster.

In some ways, *Computer Security* is a perfect book for me – that is, someone interested in security but not involved in the field professionally. Matt has produced 1,100 pages that will most likely be the "standard" for a number of years. If there are topics omitted from this book, I haven't located them. From "Access Controls" to "Policies," "Cryptography" and "Malice" to "Auditing" and well over a dozen other topics, Matt has been there and done that. He has called upon Elisabeth Sullivan to contribute a section on "Assurance," and he concludes with chapters on "Network Security" and "System Security." Even if Matt had not thanked me in his acknowledgments, I would think this is an important and masterful tome.

The 1994 book on firewalls by Bellovin and Cheswick is in Bishop's bibliography; several of Bishop's papers are in the new edition of that book, now Cheswick, Bellovin, and Rubin. When I reviewed the first edition, I was full of praise. The book is now nearly 150% of what it was eight years ago. The new edition comprises 19 chapters in six sections. The fourth ("Firewalls and VPNs"), fifth ("Protecting an Organization"), and sixth ("Lessons Learned") are worth the price of admission. The introduction to cryptography in the first appendix is excellent.

Potter and Fleck's volume on 802.11 security failed to reassure me. I still fear that we've not reconciled the convenience of wireless with confidentiality. But it's a solid presentation of the topic.

Knowledge

The concept of "knowledge management" was extremely popular in the '90s. Typically, it referred to the ways in which organizations could manage their intellectual property. This meant that "knowledge" was some sort of bankable resource ready for transfer and reuse.

More recently, as demonstrated by *Sharing Expertise*'s very title, knowledge management has taken on aspects of humanity – recognizing the human components of knowledge work and sharing, rather than merely storage and retrieval.

This anthology, a truly interesting one, exemplifies this approach.

Networking

If you work on or with the Internet, you work with DNS, BIND, and BGP. Liu's *DNS & BIND* lives next to my computer. His "cookbook" will live on my reference/how-to shelf. I found "How to prevent Windows computers from trying to update your zones" (pp. 106ff.) useful just yesterday.

Iljitsch van Beijnum's *BGP* is the best thing I've looked at on the subject since

Stewart's booklet five years ago. Really useful.

A Handful of Useful Pocket References

O'Reilly began producing pocket references a few years ago. I find many of them very handy. Three recent and valuable additions are those on C, PHP, and system administration. I think they're a bargain at about $15 each. I also found the *Word Pocket Guide* very useful (my spouse uses Word at work and I was clueless until Glenn's book came my way).

book reviews

CRACKPROOF YOUR SOFTWARE: PROTECT YOUR SOFTWARE AGAINST CRACKERS

Pavol Cerven

San Francisco, CA: No Starch Press, 2002. Pp. 250. ISBN 1-886-41179-4.

Reviewed by Jennifer Davis
sigje@sigje.org

At first glance *Crackproof Your Software* appears to be an enlightening book on a subject that hasn't been sufficiently covered. Although security is a big topic, most books cover securing your software against attacks on the software itself and on users of the software, data, and other systems on the network that the software is running on. This book hopes to illustrate how to prevent users from cracking the protection of the licensing code on your software.

As I began to read, however, I was immediately disappointed by several assumptions made by the author:

- Commercial software is written only on the Windows platform.
- Visual Basic, Delphi, and assembly language are the only programming languages commercial software is written in.
- Readers do not know what decompilers or debuggers are, but they know how to use them.

Other disappointments included useless diagrams and snapshots, like the 1/4-page size image of a window saying "Please insert the Half Life CD"; descriptions of good and bad securing applications without detailed explanation why a developer should use one and not the other; repetitive comments ("combine protection"); and overemphasis on the cracker's point of view to the detriment of the application developer's perspective in securing applications from cracking.

This book would be more aptly titled "How Crackers Crack Software on the Windows Platform."

Sadly, this book doesn't introduce the idea that future processors might have better capacity for secure programming with such new technologies as the upcoming Transmeta Crusoe chip.

My greatest dissatisfaction with this book may be that I'm not the intended audience. Someone with a great deal of assembler experience and Windows programming might be able to look at the available assembler code snippets and better understand what Mr. Cerven was trying to communicate with this book.

I look forward to seeing alternate coverage of this topic, or a better edited version of this book, since crackproofing software remains an interesting but underexamined subject.

EXTENDING AND EMBEDDING PERL

Tim Jenness and Simon Cozens

Greenwich, CT: Manning Publications, 2002. Pp. 361. ISBN 1-930-110082-0.

Reviewed by Raymond M. Schneider
ray@securityfoo.net

Ever found yourself in need of some added functionality in Perl? Ever found yourself in need of an embedded language in your application? *Extending and Embedding Perl* attempts to help the reader with just those situations.

Extending and Embedding Perl, like most technical books, starts off by gearing the reader up with the necessities for understanding the material covered.

The first three chapters are introductory, covering the very basics of the C programming language, the basics of XS (eXternal Subroutines), and more advanced C programming. The experienced C programmer may safely, in my humble opinion, "raid" the first three chapters for any bits that they are either unfamiliar with or in need of brief review.

Chapter 4, "Perl's Variable Types," covers things like how scalar variables map to C structures under the covers. It provides a nice overview that will help the reader with all the typedefs that point to C structures. On top of that, Perl wizards will be happy to find more "magic" in this chapter as the "magic variables" SvPVMG are covered here.

Midway through the text, the authors introduce the reader to the Perl API. Chapter 5 contains an incredible amount of information, with plenty of example code followed by numbered sections that step through the code and provide explanations. The reader will also find "tip" and "note" sections throughout.

The next chapter builds on the basics of XS and even discusses linking to Fortran or C++.

Now that the reader has learned all about XS, the authors present alternatives. As is common with everything relating to Perl, "there is more than one way to do it." The reader is introduced to h2xs, SWIG, and the Inline module in an effort to ease the use of XS.

At this point the book makes the transition to embedding, Chapter 8 talking specifically about embedding Perl in C applications, and Chapter 9 covering an example of embedding Perl into an application many of *;login:*'s readers are probably familiar with – Mutt.

The last two chapters of the book cover Perl internals and Perl development as the authors encourage the reader to participate in the future of the Perl language.

There is more to *Extending and Embedding Perl* than what the reader gets in the bound book. There is a Web site for the book: *http://www.manning.com/jenness*. There, in an area called "Author Online," readers may interact with the authors in a sort of question and answer situation. It's a quite interesting idea. I signed up for it and posted a question. I have to admit that after having to wait a month and a half for a response I consider the online forum a bit of a flop, especially if the reader's need for a response is in any way urgent. In a month and a half most people will have long since moved on, especially in our industry.

I can, however, recommend this book to anyone interested in extending or embedding Perl. It is a quick read that the professional programmer can devour and the novice can understand.

USENIX and SAGE Need You

People often ask how they can contribute. Here is a list of tasks for which we hope to find volunteers.

The SAGEwire and SAGEweb staff are seeking:

- Interview candidates
- Short article contributors (see *http://sagewire.sage.org*)
- White paper contributors for topics like these:

Back-ups	Emerging technology	Privacy
Career development	User education/training	Product round-ups
Certification	Ethics	SAGEwire
Consulting	Great new products	Scaling
Culture	Group tools	Scripting
Databases	Networking	Security implementation
Displays	New challenges	Standards
Email	Performance analysis	Storage
Education	Politics and the sysadmin	Tools, system

- Local user groups: If you have a local user group affiliated (or wishing to affiliate) with USENIX and/or SAGE, please email the particulars to *kolstad@sage.org* so they can be posted on the Web site.

;login: always needs conference summarizers for USENIX conferences. Contact Alain Hénon, *ah@usenix.org,* if you'd like to help.

The Decline of Research: Commentary and Book Review

by Peter H. Salus

USENIX Historian

peter@usenix.org

In March 1978, Bill Joy offered the first "BSD" tape; in September 1983, BSD 4.3 was (finally) released; in June 1993, it was BSD 4.4's "turn." The "B" tells it all: They all emanated from Berkeley. But UNIX came out of Murray Hill, New Jersey.

Let me start with the tale of several research labs whose results have had a profound effect on the world as we know it.

In 1925, AT&T and Western Electric set up a separate company called Bell Labs. Later in the 1920s, T.J. Watson Sr. set up a "Research Lab" in a brownstone near Columbia University. (In case you don't know, T.J. Watson was the founder of IBM.)

Both Bell Labs and Watson's Lab flourished for 60 years. The divesture of 1984 made Bell Labs a part of AT&T, and the breakup of 1996 caused a major fissure, with the Labs becoming a part of Lucent, but several groups are now part of the "new" AT&T.

During this same period, IBM Research first moved to Yorktown Heights, New York, expanded to the nearby town of Hawthorne, and then shrank dramatically (with falling share prices) in the 1990s.

DEC's labs grew, blossomed, and then nearly died between 1960 and 1995. The devastation wrought first by the Compaq merger and by the subsequent HP merger has been catastrophic.

The Stanford Research Lab, now transmogrified into SRI International, is still functioning, though it is very different from what it was in the 1960s; BBN is, effectively, no more. Xerox PARC has become an independent corporation, and it has shifted its focus from technology to content.

In effect, IBM's facilities currently constitute the sole functioning corporate research lab in North America (NEC Research Institute seems to be confined to computing and communications systems).

These thoughts were initiated by the arrival of two books, one by Narain Gehani, who was at Bell Labs from 1978 to 2001, the other by Severo Ornstein, one of the first engineers to work on the ARPANET at BBN.

Reading them is an exciting experience: Gehani started in the group producing Programmer's Workbench – PWB UNIX – and went on to become a "distinguished member of staff"; head of the Database Systems Department; vice president; and research vice president of Communications Software Research.

Ornstein, the son of a virtuoso pianist and major composer, had a checkered career but went to BBN in the late 1960s, where he was Frank Heart's hardware ace. He went from BBN to the West Coast, was fundamental in the formation of CPSR, and has remained active on "social issues." He still fears the control of technology by a small political and military cadre.

With this overlong prelude, let me turn to the actual books.

Gehani tells a good tale and makes a number of important points, but I think the crux of the entire book comes at the point where Shamim Naqvi is appointed research vice president of center 1138 and compensation in research is reoriented to reward those whose work is "useful for the Lucent business," as opposed to focusing on science and the publication of papers.

Remember, this is the place that gave us both the transistor and the radio telescope, just to name two of its thousands of inventions and patents. It's the place where George Stibitz built his calculator. And it's the place that created UNIX, C, C++, troff, awk, Plan9, Inferno, etc., etc. Gehani recounts the efforts of Penzias as gradually moving the Labs "away from science."

It's sad, but it's true.

I happen to believe very strongly that real research is vital. It doesn't matter what it is. We hardly ever know what's important and what isn't. Who would have dreamt that 40 years on that piece of putty with some wires sticking out of it would have been transmogrified into the board with tens of thousands of circuits on it?

If you want to understand how crass mercantilism has changed true research for the worse, you must read Gehani. Modern mercantilism corresponds precisely with a reshaping of almost all of US industry in a manner that MBAs would heartily approve of. The old "seed corn" argument comes to mind.

Ornstein's book reminds me of just where that seed corn came from. BBN was founded over half a century ago. Starting in about 1994–95, it began to wane. The magic of Ornstein and his piano, of juggling in the hallways, of "doing" acoustics and computer science and inventing the ARPANET gave way to a far more businesslike culture. After nearly a decade in decline, the morale of BBN may be improving: Someone at Verizon seems to have learned just what a gem it had, and what had happened to it.

A culture does not spring up overnight, but it can be destroyed rather quickly.

BELL LABS:
LIFE IN THE CROWN JEWEL
NARAIN GEHANI
Summit, NJ: Silicon Press, 2002. Pp. 258.
ISBN 0-929-30627-9.

COMPUTING IN THE MIDDLE AGES:
A VIEW FROM THE TRENCHES
SEVERO ORNSTEIN
N.p.: 1st Books, 2002. Pp. 275.
ISBN 1-403-31517-5.

USENIX Pledges $35,000 Matching Fund for Advancing OpenAFS

by Ellie Young

Executive Director

ellie@usenix.org

USENIX has pleged a $35,000 matching fund donation to the OpenAFS project, an ongoing collaborative effort chartered with enhancing AFS, a widely used distributed file system.

The USENIX donation, matched evenly by Intel and Morgan Stanley, will be distributed to the OpenAFS Advisory Council, which is responsible for the overall direction of the OpenAFS project. The OpenAFS Advisory Council is comprised of representatives from Carnegie Mellon University, MIT, the University of Michigan, Intel Corporation, Morgan Stanley, and IBM Corporation.

"The support from USENIX will enable the Advisory Council to devote the required time to push forward the research boundaries of AFS," said Peter Honeyman, Scientific Director of CITI, University of Michigan. "The Council will initially concentrate on bringing AFS up to date with modern transport and security, the first steps in fulfilling the promise of AFS as a high-quality, open source, wide-area distributed file system."

Pioneered at Carnegie Mellon University and supported and developed as a product by IBM Corporation, AFS offers mid-sized businesses, large enterprises, and universities a scalable, high-performance, reliable, and secure file-sharing system.

"USENIX has a long and proud history of supporting technical development," said Ellie Young, Executive Director of the USENIX Association. "Our donation to the OpenAFS project expresses our commitment to supporting important technologies that offer potential benefits to the entire computing community."

SAGE news

SAGE Update

by John Sellens,
SAGE Secretary

John Sellens is the General Manager for Certainty Solutions (formerly GNAC) in Canada, based in Toronto, a long-time system and network administrator, SAGE booklet author, and he is proud to be husband to one and father to three.

jsellens@sage.org

The SAGE Executive Committee met in Chicago on February 1 for a very productive meeting. New SAGE Exec members took office, and the Committee officers were selected (as listed below). Departing Committee members Tim Gassaway and Josh Simon were thanked for their service and contributions.

The Exec also reviewed the current status of SAGE projects, in addition to holding a productive brainstorming session for plans, programs, and strategies for the coming year. The Exec paid special attention to optimizing member services and projects in addition to publicizing opportunities for SAGE members to volunteer for projects that interest them. Expect to see results of this session in the coming months.

This year's SAGE budget is very tight, reflecting the general downturn in the tech economy that we've all experienced. The 2002 SAGE Committee, working with its Executive Director, Rob Kolstad, and the USENIX staff, pared the budget for this year, ensuring that we don't overstep our financial reach. This means, however, that many of our programs have been scaled back, and some have been put on indefinite hold. We are, however, working on a number of fronts to increase our visibility, membership, and fund-raising, all of which will help to improve our financial position and ability to offer services to our members.

One of the programs hit hardest by the financial cutbacks has, unfortunately, been the SAGE certification program. The existing contract with the testing provider was allowed to expire in January, and we are currently considering alternative test delivery plans that will enable us to continue offering the program in a more cost-effective manner.

In early February, a draft SAGE Code of Ethics was posted on SAGEwire for public comment. The results will be presented to the SAGE Committee at one of its weekly telephone meetings for final approval. Look for the results of this process by April.

Rob Kolstad has been updating and enhancing the annual SAGE Salary Survey and Sysadmin Work Profile, which should be available online by the time you read this. The salary survey has always been a very useful tool for sysadmins, and we encourage you to participate and to spread the word to your colleagues – the more the merrier (and the better the resulting data)!

The LISA 2003 Call for Participation has been issued with a submission deadline of April 21 (see it online at *http://www.usenix.org/lisa03/*). This year's conference will be in late October in sunny San Diego. Conference chair Æleen Frisch and the rest of the program committee are sure to put together an outstanding program. Get involved and help make this year's LISA program the best ever!

Finally, I must mention that the SAGE Committee exists to serve SAGE members and to work to provide programs and resources that reflect your needs and interests. We would love to hear from you as to your interests and your suggestions for programs. If you're interested in getting involved in SAGE programs and activities, many opportunities exist for you – just let us know that you're interested in being involved. We welcome your feedback to the SAGE Committee (*sage-exec@sage.org*) or to individual members of the Committee.

New officers:

President: Geoff Halprin

Vice-President: Trey Harris

Secretary: John Sellens

Treasurer: Bryan Andregg

Member: Gabe Krabbe

Member: David Parter

Member: Peg Schafer

conference reports

This issue's reports focus on WIESS '02 and on OSDI '02.

OUR THANKS TO THE SUMMARIZERS:

Scott Banachowski
Richard S. Cox
Steven Czerwinski
Himanshu Raj
Cristian Tapus
Charles P. Wright
Praveen Yalagandula
Wanghong Yuan
Nickolai Zeldovich
Ben Zhao
Yutao Zhong

2nd Workshop on Industrial Experiences with Systems Software (WIESS '02)

BOSTON, MASSACHUSSETS
DECEMBER 9-11, 2002

KEYNOTE ADDRESS

Douglass J. Wilson, IBM

Summarized by Richard S. Cox

MIT's *Technology Review* recently ran a story titled "Why Software Is So Bad."

The key is the problem of integration. CIOs spend 35% of their budgets on integration, because every new system must work with the existing infrastructure. The complexity of integration is driven up by the constraints of the business environment as well as those of the software.

Several lessons can be learned from studying systems usage. First, standards and componentization are proving ineffectual for complex systems. For example, LDAP is a fine protocol, but no two organizations use the same schema. Making matters worse, interoperability is poor due to differing interpretations of standards, edge conditions, and vendor-specific extensions. This is leading to a change from creating solutions by mixing "best-of-breed" products to using a single "best-of-suite" package. Unfortunately, much of the literature on building component systems is academic, failing to deal with the scale of large systems.

Second, systems will fail. Other industries have accepted this, but software engineers are just now realizing that failure is hard. The recovery design must fit the usage, which means the designer must understand the failure modes in practice. This may mean using less sophisticated algorithms that are better fitted to the purpose. It also means accepting that business redundancy may be at odds with IT redundancy. For example, a stock trading system may have several redundant pathways for entering a trade, to protect against trades being lost before they have been entered. For the IT infrastructure, this means the redundant pathways need to be synchronized at some point. This type of problem is rarely considered by researchers or product developers.

Third, error logging and reporting is important. As an industry, we currently support very primitive logging with no mechanisms for root-cause analysis or correlation of failures. Error messages are often arcane or not useful, and "first-failure" capture is impossible. This is evidenced by a common, though unrealistic, request from support center staff: "Turn logging on and recreate the failure." Because logging events need to be correlated, error tracking and logging should be a basic service of the OS.

SESSION 2

Summarized by Wanghong Yuan

USING END-USER LATENCY TO MANAGE INTERNET INFRASTRUCTURE

Bradley Chen, Michael Perkowitz, Appliant

The problem addressed in this paper is that distributed application performance is important but hard to understand. CDN selection and CRM systems were offered as examples to illustrate the problem. The basic approach proposed is to use end-user latency analysis: (1) content (e.g., an HTML Web page) is tagged to collect data; (2) tagged data is observed on the desktop (end-client system); and (3) data is analyzed on the management server.

The challenges for this approach include (1) technique issues such as larger data sets, heavy-tailed data, and the derivation of request properties, and (2) social and economic issues such as privacy. The results show that end-user latency analysis can monitor relevant information, which is obscured otherwise.

Building an "Impossible" Verifier on a Java Card

Damien Deville, Gilles Grimaud, Université de Science et Technologies de Lille

The smart card device environment imposes constraints on CPU, memory, and I/O. As a result, Java Card Virtual Machine needs to be adapted to the smart card. The regular verification approaches do not fit, since unification is costly. The proposed approach addresses the above problems via (1) non-stressing encoding and (2) efficient fixed points using a software cache policy.

Enhancements for Hyper-Threading Technology in the Operating System: Seeking the Optimal Scheduling

Jun Nakajima, Venkatesh Pallipadi, Intel

In this talk, Jun Nakajima first gave an overview of Hyper-Threading (HT) technology by comparing it with multiprocessors. The reason behind HT is that CPU units are not fully utilized. To fully utilize CPU units, the HT approach is to use two architectural sets, thereby executing two tasks simultaneously.

The HT approach requires the OS scheduler to support HT-aware idle handling, processor-cache affinity, and scalability (per-package run queue). This paper proposes a micro-architecture scheduling assist (MASA) methodology to address the above problems, thereby achieving an optimal process placement.

INVITED TALK

Software Strategy from the "1980 Time Capsule"

John R. Mashey

Summarized by Yutao Zhong

John Mashey reused the slides from a talk he gave 25 years ago titled "Small Is Beautiful and Other Thoughts on Programming Strategies." It is interesting to see from these old slides and the newly added comments what has changed and what hasn't.

The previous talk was given in 1977, when the main computer models were IBM mainframes, coming VAX, and PDP-11s, while C was taking the place of ASM and structured programming became the dominating idea. Three approaches of building system software were introduced and compared: "Do it right," "Do it over," and "Do it small, with tools." "Do it right" emphasizes an optimistic on-requirement analysis that assumes "we know what we are doing." "Do it over" puts more emphasis on early implementation by still starts from scratch. The last approach, by contrast, considers tools instead of systems and builds small and fast so that, if necessary, failures can happen quickly.

In order to see the effect of these strategies, Mashey discussed different metrics to qualitatively measure success and gave statistics and observations of projects in data processing. Figures and numbers showed the low percentage of complete success and indicated the larger a project is, the higher overhead it has to pay. Laws of program evolution also state that the entropy of a project increases with time and may result in a complex program used to solve a simple problem.

Several principles were offered to counteract these problems: "build it fast," "keep it small and simple," and "build for change." Existing tools should be utilized whenever possible. It would be good to build tools and consider the interfaces of connecting tools. Some "small tactics," including "lifeboat theory," "sinking lifeboat theory," and other considerations about people and consolidations, were also discussed.

Even after 25 years of work, we need to keep these problems in mind, since system complexity is much higher nowadays; fortunately, people are increasingly aware of these issues.

Mashey ended the talk by saying, "We have met the enemy and they are us."

SESSION 4

Summarized by Cristian Tapus

An Examination of the Transition of the Arjuna Distributed Transaction Processing Software from Research to Products

M.C. Little, HP–Arjuna Labs; S.K. Shrivastava, Newcastle University

Arjuna started in 1986 as a research project at the University of Newcastle, England. Arjuna was a "vehicle for getting Ph.D. degrees." The decision to use C++ was a pragmatic one (expensive Euclid vs. free C++ AT&T). Arjuna was designed to be a toolkit for development of fault-tolerant applications which would provide persistence, concurrency control, and replication. Modularity was the key to the longevity of the system.

In 1994 Newcastle University asked them to implement a student registration system because the "academic researchers are cheap." The system was supposed to run on multiple platforms, serve about 15,000 students over five days, and could not tolerate failures. There were problems, though. Assumptions were made about network partitions and recovery that made the system fail to identify dead machines vs. slow connections. Intuition is not a good approach to designing systems.

The year 1995 brought standards for transactions: object transaction system specifications (OTSS) from OGM. It shared many similarities with Arjuna, but it was only a two-phase commit protocol engine (persistence and concurrent control where required from elsewhere). At this time the OTSArjuna system was developed. With only slight changes to the interfaces between modules, the system was complying with OTS. JTSArjuna followed just two years later as the first Java transaction service.

In 1999 the Java and C++ transaction service were marketed; only one year later Bluestone took over Arjuna Solutions Limited and was, in turn, acquired by HP in 2001. When the system was acquired by Bluestone the need for real testing became a reality. For the previous decade only about 20 tests had been used, but this was increased to over 4000 tests in order to stretch every feature of the system. The previous method was to get a release out to the users, and users would then report problems back and bugs would be fixed. Not anymore. The industry method was different from many perspectives: write manuals and white papers and train other people. "I used to laugh at white papers, but I realized you need skills. An academic person cannot do it. Academic people do technical reports, which are different," Little said.

Was it worth it? YES. But it was stressful moving away from R&D. "If you have a family don't do it. If you are in the industry and you feel you are stressed up, move to academia."

When asked what they would do differently, the reply was that they would (1) get somebody else to do the failure recovery and (2) make sure that they would have more than 20 tests.

Tree Houses and Real Houses: Research and Commercial Software

Susan LoVerso and Margo Seltzer, Sleepycat Software

Susan talked about the process they followed to make a commercial product out of BerkeleyDB. The main argument was that a research prototype is like a tree house – it doesn't last – while a commercial product is the real house. Sleepycat was founded in 1996 and transformed DB1.85 into a real product. It added transactions, utilities, and recoverability while continuing to be open source. Sleepycat is a "distributed" company; with employees spread across the world, it is hard for them to interact with each other directly. But the heterogeneous environment makes the company more powerful. In order to produce quality software, however, Sleepycat must follow rigorous software practices. Designs and reviews are sent to the entire engineering staff (one advantage of being small), and there are strict coding standards (it is the *law*).

The talk continued by describing techniques used to obtain the final product. When you hit bedrock, try to rethink what you are doing; and observe the "rule of holes – if you are in one, stop digging." In the end, certain lessons were learned from the development process: Designs and reviews are important, but reviews are not perfect; there needs to be a willingness to stop and change course when necessary and to throw code away, even if it works; and you need someone nearby who's close to the process but objective.

JOINT WIESS/OSDI PANEL

Research Meets Industry

Chair: Noah Mendelsohn; Panelists: Ramon Caceres, ShieldIP; Mark Day, Cisco Systems; Charles Leiserson, MIT; Dick Flower, HP; Brian Bershad, University of Washington

Summarized by Nickolai Zeldovich

The joint panel discussed issues related to the ridge between research and industrial development and their correlation. The discussion was entitled "Research Meets Industry."

Brian Bershad: Knowing how to teach helps in the industry, as does having a degree. You need to know how to manage and motivate people in academia. Coming back to academia, you start asking questions like, "Who cares about this project?" "Will it scale?" and so on.

Below are capsules of the discussion by the panel and the audience

Someone Whose Name I Forgot: People in academia are generally not interested in details, testing, and usability, which are needed to take something from research to a product. The industry in general is also not very interested in research work, reading papers, and so on.

Mark Day: More incentives are needed to get industry and academia to interact. Currently there are almost no such incentives.

Andrew Hume: I do technology transfer at AT&T. The problem is enticing researchers, because you go for a while without publishing papers. On the other hand, you can then write a different kind of paper, about the real aspects of systems. Academia should care more about results having to do with real details.

Noah Mendelsohn: Academic papers don't line up with industrial interests. If conferences did accept industry papers, will companies write them?

Andrew Hume: Yes. Motivating factors are satisfaction and recognition, perhaps because this is rare.

Noah Mendelsohn: There's also an opportunity cost to writing a paper, of losing developer time.

Charles Leiserson: Students going into industry don't understand company culture; they are used to the academic environment.

Margo Seltzer: There needs to be motivation for companies to write papers. Engineers want to write papers, but need to sell papers to managers, as a tool for marketing, for example.

Brian Bershad: Often companies don't want intellectual property published. Thus, commercial papers lack technical detail.

Charles Leiserson: Writing papers is usually as useful internally as getting it published. Papers help internal communication.

Roblis(?), Intel: At Intel Labs, writing papers is rewarded and expected. In the product groups, however, it is viewed as a net negative. It would be useful if conferences could accept/reject rough drafts to avoid wasted write-up efforts.

Anthropologists studied engineers and found that usually there are a few "leaders" in engineering groups that go to conferences, lead things to turn something into a product, etc. Maybe we don't need people transfer, we just need to market things to these "leaders"?

Dick Flower(?): There are groups without leading individuals. Having an advanced development group of some sort could be useful, though.

Brian Bershad: I think some companies have reasonable expectations of the research world, and some companies don't.

John(?): The HotChips conference, for example, only produces presentations and not papers. It's much easier to get a presentation, rather than a paper, from a lead chip designer.

Mark T (MS Research): At PARC, of the people who went to industry, none ever came back for long. Can you ever come back from the industry?

Brian Bershad: No, not possible to come back and be the same. Your focus changes to short-term goals.

Charles Leiserson: In my lab, lots of people, including staff, did it OK. Doing so colors your interest, though. You learn about things like barriers to adoption, etc.

Mark Day: Do you mean returning to applied research or to academia?

Mark T (MS Research): PARC returnees were successful in the industry and kept going back to form new startups. Focus on doing something with impact in the world.

Noah Mendelsohn: Having gone to industry before grad school gave me a great perspective on reality, judging the realism of projects, etc. It's very hard to do research part-time.

Bradley Chen (Appliant): What do you think about requiring faculty to have industrial experience?

Charles Leiserson: Depends on the quality of the experience. But yes, there are things from industry to be taught to university people. Management, leadership, motivation, educating about teamwork, working with each other.

Fred Douglis (IBM): Were there more core industrial papers back when USENIX took extended abstracts?

Chris Small: We seem to have a hangover after the dot-com boom. There was a huge flux of ideas from research to commerce.

Brian Bershad: The dot-com boom shows what happens when the barriers to adoption from research are removed. The result wasn't so great – too many worthless ideas, no industrial experience. Doing a startup is easier the second time around.

Someone from VMware: We were lucky to have good timing to submit our paper to OSDI – the submission deadline was a few months after an internal deadline, which gave us time to gather results. The community should be more receptive to papers about released or dead products; they are valuable.

Jun Nakajima (Intel): In this economy, R&D costs are being reduced and moved to China. For the cost of one engineer here you can get three to five engineers in China. How do you justify the three-to-five times cost?

Charles Leiserson: Education. Also location – most other companies are located here.

Erez Zadok (Stony Brook): Academia is not preparing students for life in industry. It's difficult to convince universities to create courses with practical aspects.

5th Symposium on Operating Systems Design and Implementation (OSDI '02)

TECHNICAL SESSIONS

DECENTRALIZED STORAGE SYSTEMS

Summarized by Himanshu Raj

FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment

Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, Roger P. Wattenhofer, Microsoft Research

The goal of this research was to make a scalable serverless distributed file system while maintaining security against malicious attacks in untrusted systems. Byzantine protocols are used to define untrusted infrastructure. The FARSITE solution is a virtual global file store of encrypted data that is replicated to facilitate availability. Storage is divided into two parts: file data and metadata. Metadata information is a hash-computed form of actual file data. The system is built around an infrastructure to store file data and metadata separately, and traditional Byzantine properties are applied only for machines storing metadata.

Since Byzantine operations are costly, they are not performed per file I/O. Instead, a Byzantine operation is defined valid for a period of lease. Various different types of leases are available to suit different consistency requirements. Batching is another concept used to reduce cost. The system is implemented as a service in user level and as a kernel mode driver that routes the actual file system calls to NTFS. According to the results reported, the system performs better than a central file system, though worse than bare-bones NTFS. The system is not designed to address efficient large-scale write sharing, database semantics, or disconnected operations. The project link is *http://research.microsoft.com/sn/Farsite/*.

Taming Aggressive Replication in the Pangaea Wide-Area File System

Yasushi Saito, Christos Karamanolis, Magnus Karlsson, Mallik Mahalingam, HP Labs

Pangaea is a scalable distributed file system targeted for the type of WAN infrastructure characteristic of multinational companies with overseas corporate offices and a need to share data. Design goals of Pangaea include hiding WAN link latencies, availability in a high-change environment, and network usage efficiency. The system assumes the presence of an available secure infrastructure, such as VPN. The system also provides eventual consistency, though manual open/close-style consistency could also be provided. The system employs pervasive replication to dynamically replicate each file/directory in the system independently. Benefits drawn from intensive replication are speed, availability, and network efficiency. The system is implemented in user space based on SFS API. The NFS client at the kernel level routes the I/O requests to the Pangaea service at the user level. The system uses graph-based replica management. A random graph is created for every file/directory in the system, and edges of this graph are used for update propagation among replicas and for replica discovery. Since the system does not have a central lock manager, it uses a technique called harbingers to compute a spanning tree so that duplicate transmissions can be avoided. This technique also helps reduce the propagation delay. The project link is *http://www.hpl.hp.com/research/ssh*.

Ivy: A Read/Write Peer-to-Peer File System

Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, Benjie Chen, MIT

The main goal of Ivy is to build a highly available file system out of inexpensive infrastructure that can scale to multiple writers on the same data. The system leverages DHT in the core, and provides weaker consistency guarantees for metadata for performance reasons. The main idea behind the system is to use a log per user and combine potentially multiple logs to serialize the updates made on a shared object. Serialization is based on a version-vectoring scheme rather than using a timestamp technique. The evaluation compares Ivy's performance with a local file system to see the load characteristics and runtime comparisons made with NFS over WAN. Results show that log operations tend to dominate the performance of the system over WAN and parallel fetching of log records can be used to hide latency. The system addresses sharing among only a small number of writers and hence does not address the scalability issues involved with a large number of writers sharing an object. Merging of logs is performed later, as in the Coda file system, and conflict resolution is addressed then. The way to provide effective read sharing in Ivy is to use multiple file systems. The project link is *http://pdos.lcs.mit.edu/ivy*.

ROBUSTNESS

Summarized by Ben Zhao

Defensive Programming: Using an Annotation Toolkit to Build DoS-Resistant Software

Xiaohu Qie, Ruoming Pang, Larry Peterson, Princeton University

Qie began by examining how typical DoS attacks work. One attacks a Web server, for example, by intentionally slowing down TCP, faking packet loss, and attempting to tie down as many TCP connections at the server end as possible. It is useful to classify resources as renewable (CPU, network bandwidth, disk bandwidth) and nonrenewable (processes, file descriptors, memory buffers). Renewable resources are vulnerable to "busy attacks," which try to request the resources faster than they can be allocated. The corresponding solution is protection via admission control. Nonrenewable resources are vulnerable to "claim-and-hold attacks," which attempt to request and hold on to them. The corresponding solution would be to recycle resources when they are exhausted, reclaiming them from certain applications. Combinations of the two types of resource attacks are harder to deal with. For example, when file descriptors (nonrenewable) are recycled to protect against claim-and-hold attacks, they become a renewable resource and therefore vulnerable to busy attacks.

The proposal is to utilize a toolkit containing "sensors" and "actuators" to protect both types of resources, with low programming burden. The toolkit is a combination of techniques from work in protection, static analysis, anomaly detection, and profiling. To protect renewable resources, the approach is to divide functionality into distinct services and balance resources among them, such that the impact of an attack on a single service is limited to that service. To protect nonrenewable resources, they need to be recycled when necessary. The algorithm to choose the resource instance to reclaim can be driven by a timer. The timer can be set on idleness or on the length of the service lifetime. The work proposes a user-defined progress metric (amount of data output or number of state transitions) that will reclaim resources from the "slowest" principal.

The toolkit is implemented as 11 C macros and library functions. The authors also modified gcc for auxiliary code generation at compile time. The evaluation contains case studies of a flash Web server. The Web server is partitioned into 46 services; 60 annotations were added to the code. Under a slash attack, the annotated server response time is 5.1 milliseconds, compared to a normal response time of 4.3 milliseconds, and is significantly lower than a non-annotated server under attack, which has a response time of 25 seconds. A possible limitation is that its effectiveness depends on service granularity. The project link is *http://www.cs.princeton.edu/nsg*.

Using Model Checking to Debug Device Firmware

Sanjeev Kumar, Kai Li, Princeton University

Device firmware is a piece of concurrent software that achieves high performance at the cost of software complexity. It contains subtle race conditions that make it difficult to debug using traditional debugging techniques. The problem is further compounded by the lack of debugging support on the devices. Model checking is a promising approach. It can systematically explore all possible scheduling orders and provide counter-examples of bugs found. The general technique is to extract models from programs either manually or via a compiler. The authors extracted models for the Spin model from programs written in the ESP language. ESP is a language for programmable devices that the compilers use to generate tests. In evaluation, the techniques are applied to VMMC (high performance communication design that bypasses the OS for data transfers). VMMC firmware was reimplemented using ESP. Seven models were found using abstract models, despite the global nature of some bugs (deadlock). These bugs would be hard to find without using a model. Where a full search of the state space is not possible, partial searches can minimize resource costs and still produce useful results.

CMC: A Pragmatic Approach to Model Checking Real Code

Madanlal Musuvathi, David Y.W. Park, Andy Chou, Dawson R. Engler, David L. Dill, Stanford University

Many system errors do not emerge unless some intricate sequence of events occurs. In practice, this means that most systems have errors that only trigger after days or weeks of execution. Model checking is an effective way to find such subtle errors. This work contributes the C model checker, which links to code, emulates a real system, captures the states of the system, and analyzes the results. CMC schedules threads to emulate nodes in the network, where scheduling granularity is on the order of entire even handlers. This means handlers are treated atomistically, and synchronization bugs can be missed. CMC tries to search the entire space, but it can checkpoint at decision points and resume later where different states can be generated. The work uses three optimizations to reduce the search space: hash compaction, downscaling, and state canonicalization. Hash compaction is the use of hashtables to store previously seen states so they are not examined again, and computing hashed signatures for each state to reduce space requirements. Downscaling is the use of a small number of nodes in order to reduce the state space. Complex interaction bugs can still be produced, but it might miss bugs only seen on large-scale interactions. State canonicalization is the simplification of similar states down to a single state, which is then evaluated. When applied to AODV routing protocol implementations, the CMC checker found 42 bugs (of which 34 are distinct, and one is a bug in the specification).

KERNELS

Summarized by Charles P. Wright

Practical, Transparent Operating System Support for Superpages

Juan Navarro, Rice University and Universidad Católica de Chile; Sitaram Iyer, Peter Druschel, and Alan Cox, Rice University

Translation lookaside buffer (TLB) coverage has decreased by a factor of 1000 in 15 years. In 1985 the TLB miss overhead was less than 5%; today it is over 30%. This is primarily due to increases in the size of working sets, yet TLB size has remained constant. Many architectures allow the creation of superpages. A superpage TLB is like a normal TLB, but a size field is added. Navarro presented a practical implementation of superpages for FreeBSD 4.3.

There are three major issues when implementing superpages: (1) superpages require allocation of contiguous, aligned memory; (2) a superpage can be created out of several normal pages (promoted) or broken into several pages (demoted); and (3) the need to prevent internal fragmentation. Each issue is dealt with in an opportunistic manner. For example, once an application touches the first page of a memory object it will quickly touch every page. Each superpage is created as long as possible and at the earliest point. To do this, reservation is employed, but it may be broken if the memory is needed (the oldest reservations are broken first). The same type of opportunistic algorithm is applied to promotion and demotion. To keep fragmentation low the page demon restores contiguity, and wired pages are clustered. On the SPEC CPU 2000 integer and floating point operations, a performance improvement of about 11% was observed. For a large matrix transposition, an improvement of over 600% was observed.

More information is available at *http://www.cs.rice.edu/~jnavarro/superpages/*.

Vertigo: Automatic Performance-Setting for Linux

Krisztián Flautner, ARM Limited; Trevor Mudge, University of Michigan

Flautner presented a software framework to do energy management by setting processor speed. The processor consumes 32% of the power budget on small devices (e.g., PDAs). Vertigo focuses on power management when the CPU is performing work, not when the CPU is idle. The underlying principle is to run just fast enough to meet deadlines, without using higher power consumptions. An increase in performance will create an exponential increase in energy usage—it is better to use a smaller amount of computing power for a longer period of time than to use a large amount of power over a short period.

Vertigo is a Linux kernel module that monitors system execution to determine how fast things need to go. There are five hooks in the kernel (e.g., task switching, some system calls, and swapping) that are used to determine activity. A policy stack combines multiple simple algorithms to determine the best performance level. Each algorithm can be specified for a specific performance situation. For example, an interactive performance algorithm may monitor X server events.

Vertigo was compared to the Crusoe LongRun on-chip power saving system. Using application-specific knowledge was very effective. For example, the Crusoe LongRun would cause spikes to full power when the GNOME clock ticked. When playing MPEG movies both Vertigo and LongRun do not drop any frames, but Vertigo used 52% of the peak performance level and LongRun used 80%. The conclusion is that the kernel has lots of valuable information that is lost on the chip.

Cooperative I/O: A Novel I/O Semantics for Energy-Aware Applications

Andreas Weissel, Bjorn Beutel, and Frank Bellosa, University of Erlangen

Traditional operating system power management assumes the timings of disk operations by user applications are unknown and cannot be influenced. Additionally, transitioning to a low-power mode will actually waste power if the transition was unnecessary. Cooperative I/O changes this assumption by introducing three new system calls: coop_read, coop_write, and coop_open. Along with the standard parameters for these calls, a timeout and an abortable flag are passed (e.g., a MPEG player may specify that I/O can be deferred until the frame actually needs to be decoded). This allows the operating system to schedule I/O intelligently.

There are three components to cooperative I/O: a modified IDE driver that shuts down the disk after the break-even point (the number of seconds required to reduce overall power consumption), VFS modifications, and ext2 modifications. The goal of these modified components is to cluster I/O operations into batches, thus leaving the drive idle for the longest period of time possible. For the Amp MP3 player, 150 lines of code were modified (the bit rate was used to determine timeouts). While this modified Amp was running, an unmodified mail client using write was used. Using coop_read, a power consumption was reduced to 210 joules from 373 joules. This is a better energy savings than an "Oracle" policy and one which always makes the right power decision based upon a previous trace, without modifying the timing of the I/O.

PHYSICAL INTERFACE

Summarized by Charles P. Wright

TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks

Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, University of California, Berkeley; Wei Hong, Intel Research

Sensor networks are a collection of small, inexpensive battery-run devices with sensors and RF interfaces. Programming a sensor network is a difficult task: It took two weeks for two experienced students to program a vehicle-tracking sensor network. TAG eliminates the need to program sensor networks by using an SQL-like declarative language—using TAG, the same vehicle position network was programmed in two minutes. Sensor networks are installed under harsh conditions (e.g., in habitat- or earthquake-monitoring applications). The primary metric used for sensor networks is power consumption. Berkeley "Mica Motes" run for only two to three days when using full power but can last up to six months at a 2% duty cycle. Communication dominates the power consumption cost, so they use bytes sent as a metric.

To reduce the communications overhead, TAG allows in-network processing of aggregate queries (count, max, average, etc.). Madden asserts that most common data-analysis operations are aggregate operations. For example, the average temperature over all the sensors (or in a given sector) is a more interesting indicator than the temperature at each individual node.

There are several methods that can be used to decrease communication. The first method is to incrementally compute values using partial state records (PSRs). For example, an average can be transmitted to a node's parent as a sum and a count, and the parent's values can be inserted into this PSR. Additionally, snooping or guesses can improve performance. If the desired aggregate is the max, a node does not need to communicate its own value if it hears a value larger than its own. If the root knows the max value is at least 50, then it can reduce communication by communicating this value to other nodes.

More information can be obtained at *http://telegraph.cs.berkeley.edu/tinydb/*.

Fine-Grained Network Time Synchronization Using Reference Broadcasts

Jeremy Elson, Lewis Girod, Deborah Estrin, UCLA

To present a consistent view of information, sensor networks need to have a consistent view of time. This problem has already been solved on the Internet (e.g., NTP), but sensor networks do not have the infrastructure available to Internet hosts. Sensor applications also have stronger time synchronization requirements than the Internet (tracking phenomena may require microsecond-level synchronization).

Elson presented reference broadcast synchronization (RBS). Traditional synchronization methods have lots of nondeterministic delay when sending packets (e.g., backoff timers or link-level retransmission). Receiving a packet that a host sent has much less variation than the time it takes to actually send a packet

(1 bit width for receive vs. 1000 for send). Therefore, two hosts can make note of the time they received a packet sent by a third host. The two receivers now know the difference between their clocks. Clock skew perturbs this observation, however, so a best-fit line is used to determine the difference.

RBS synchronized the clock on a Compaq iPaq to precisions of 6 microseconds, whereas NTP is only able to obtain a precision of 53 microseconds. The clock resolution on the Linux platform is only 1 microsecond; Elson believes that a more accurate clock would yield better results. The performance under a 6 Mbps load shows even better results: RBS degrades to 8 microseconds, but NTP degrades to 1542 microseconds.

RBS effectively removes sender nondeterminism from network time synchronization. This facilitates a wide range of applications, including acoustic ranging and collaborative signal detection.

Supporting Time-Sensitive Applications on a Commodity OS

Ashvin Goel, Luca Abeni, Charles Krasic, Jim Snow, Jonathan Walpole, Oregon Graduate Institute

Fast processors enable interactive real-time applications in software: for example, software radio, software modems, voice over IP, video conferencing, and accurate network traffic generators. However, these applications need millisecond to microsecond timing guarantees. It has long been accepted that to provide such timing guarantees a special real-time OS is required and that general purpose OSes need a complete redesign. Real-time operating systems have many disadvantages (e.g., nonstandard interfaces and small user communities). Goel presented time-sensitive Linux (TSL). TSL aims to provide real-time performance on commodity general-purpose operating systems using an evolutionary approach.

The requirements for TSL were fine-grained timers, a responsive kernel, and an accurate implementation of a good scheduler. There are two types of kernel timers: fine-grained (soft) or one-shot timers (firm). There are two overheads when evaluating timers: reprogramming and interrupts. Reprogramming the timer turns out to be inexpensive, but the interrupts are expensive. However, soft timers have a potentially unbounded latency. TSL uses firm timers. Firm timers insert checks into kernel paths (e.g., system call entry/exit to check the timer) but also use one-shot timers that are configured to overshoot the required delay. This provides some guarantee while, hopefully, reducing the number of interrupts.

Linux has already broken down the big kernel lock, but some locks are still large. TSL uses voluntary lock yielding to increase kernel responsiveness. Finally, TSL implements a proportional share scheduler that provides a constant-speed virtual machine. Even heavy-load software modems (which require 4–16 millisecond guarantees) are supported. TSL implements sub-millisecond-timing guarantees on a general-purpose operating system. TSL imposes 1.5% overhead, which is low for firm timers. For the additional kernel preemption points, an overhead of 0.5% was introduced.

PANEL

Summarized by Steven Czerwinski

Self-Organizing Networks from Sensor Nets to P2P: Panacea or Pipe-Dream?

Co-Moderators: Peter Druschel, Rice University; David Culler, University of California, Berkeley/Intel; Panelists: Hari Balakrishnan, MIT; Yaneer Bar-Yam, NECSI; John D. Kubiatowicz, University of California, Berkeley

Are self-organized networks superior to traditionally engineered solutions and are they necessary to solve today's problems? In Culler's opinion, self-organized networks are necessary but require engineering in order to build systems with the desired predictable global behaviors. They are necessary in sensor networks because of the near impossibility of administering and configuring millions of sensor nodes; they must be able to self-organize. However, as he showed with several different self-organizing methods to compute spanning trees, designing the local rules that lead to correct global behavior is difficult. This is where engineering needs to be applied.

Balakrishnan also advocated self-organized networks because they can eliminate human misconfiguration from distributed systems and allow such systems to adapt to errors and change. He argued that distributed systems are all about enabling autonomy at subsystems, but with this autonomy come problems with misconfiguration. Using traces, he showed how a significant portion of invalid DNS queries were caused by human misconfiguration.

Kubiatowicz used a thermodynamics analogy to argue the importance of self-organizing networks. Large systems can exhibit stability through statistics if they possess replicated components that interact and adjust to one another. Energy could be injected into the system through both passive and active correction mechanisms. He labeled such systems as "thermospective." With Moore's Law enabling redundancy and with the need to eliminate human configuration, he saw these systems as being the future.

Druschel presented a spectrum of current distributed systems, with decentralized approaches on one end and self-organizing ones on the other. He argued that natural (biological) systems are the only truly self-organizing systems, with sensor networks being fairly close. Systems requiring ACID semantics have difficulties making it onto the self-organized end. He also noted that the systems we engineer are robust to both mundane failures and malicious attacks, while self-organizing ones are only robust to mundane failures. They would require (at the least) a trusted certificate authority to be robust to attacks.

Bar-Yam used analogies from biology to show that we already have the conceptual tools to demystify self-organizing networks. It may be hard to understand the progression of a mouse embryo from a macroscopic perspective, but that's because we don't understand the local rules or patterns of behavior of the smaller components. He showed how different types of patterns of behavior (such as local majority, two-dimensional condensation, and local activation/long-range inhibition) can lead to interesting phenomena, such as the stripes on a zebra's back.

Audience members pointed out the difficulties of creating such a system from an economic and business standpoint (who pays for all of this?) along with privacy concerns (do you really want your data going anywhere and everywhere?). Some also cautioned against the misuse of biology and other non-computer science metaphors, which can encourage similarities being drawn where none exist.

VIRTUAL MACHINES

Summarized by Praveen Yalagandula

Memory Resource Management in VMware ESX Server

Carl A. Waldspurger, VMware

This won the Best Paper award.

VMware ESX server is a thin kernel to multiplex hardware resources among virtual machines. The three main issues that arise in memory resource management are fairness, performance isolation among virtual machines, and efficient utilization of the available machine memory.

To efficiently reclaim memory from a virtual machine, Waldspurger proposes the ballooning technique where a driver inside the virtual machine allocates some pages, forcing the guest OS to evict pages not in use or to swap some pages. Experimental results show that there is only a small overhead of 1.4% to 4.4% in using this technique.

Efficient use of available machine memory is provided through memory sharing, where a single page on the machine is shared by multiple VMs (using copy-on-write semantics). A background process computes hashes of pages to determine the duplicate pages. For "best case" workloads, in which multiple Linux VMs are run, about 60% memory savings are observed. For real workloads, the savings ranged from 7% to 32%.

For a memory allocation scheme that provides fairness among virtual machines while being efficient, the author proposes the concept of "idle memory tax," where the idle pages are charged more than active pages. This new mechanism resulted in a 30% throughput increase for the workload considered in experiments.

Scale and Performance in the Denali Isolation Kernel

Andrew Whitaker, Marianne Shaw, and Steven D. Gribble, University of Washington

The goal of this work is to enable the execution of untrusted code while providing isolation so that the untrusted code does not interfere with any other process on the system. The Denali "isolation kernel" isolates untrusted software services in separate protection domains. The approach is to use virtual machines to provide isolation with strategic modifications for scalability, simplicity, and performance.

Denali's virtual machine architecture achieves scalability and performance at the cost of giving up backwards compatibility. It omits rarely used features like BIOS, protection rings, etc.; revises interrupts and MMU; and simplifies hardware I/O instructions. The resulting core kernel is an order of magnitude smaller than the bare-bones Linux 2.4.16 kernel.

For scalability, Denali employs the following techniques: (1) batched, asynchronous interrupts – instead of invoking a VM when interrupt arrives, interrupts are batched together and applied when the corresponding VM is scheduled, thus reducing the overhead of context switches; and (2) idle-with-timeout instruction – this instruction allows VM to specify how long it yields, thus leading to better scheduling. The first technique provided a 30% improvement in performance in experiments, and the second scheme yielded a 100% throughput improvement.

ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay

George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen, University of Michigan

The aim of this work is to provide a way for post-mortem analysis of intrusions. Typical system logs are subverted by the intruder. The "CoVirt" project aims at enhancing security by running the target OS and all target services inside a Virtual Machine (VM) and then adding security services in the VM or host platform. ReVirt aims at checkpointing and logging a VM's execution trace so that it can be replayed later. The virtual machine used is UMLinux, a Linux kernel that can be run on any other Linux machine.

To enable complete replay, checkpointing is done that covers the memory, CPU, and disk states, and logging is done that covers all keyboard, network events, and interrupts, along with the data corresponding to these events. Replaying the interrupts is a hard problem, and the authors use tuple, as in the Hypervisor project, to uniquely identify the place in execution where interrupts should happen.

The virtualization overhead ranged from 1% to 58% for different workloads. The logging overhead on runtime is about 8%, and the log grew at a rate of 1.4GB/day in the worst case workload and at 0.04GB/day in the best case.

CLUSTER RESOURCE MANAGEMENT

Summarized by Praveen Yalagandula

INTEGRATED RESOURCE MANAGEMENT FOR CLUSTER-BASED INTERNET SERVICES

Kai Shen, University of Rochester; Hong Tang,University of California, Santa Barbara; Tao Yang, University of California, Santa Barbara and Ask Jeeves; Lingkun Chu, University of California, Santa Barbara

The challenges involved in hosting large-scale resource-intensive Internet services on a server cluster are: (1) scalability and robustness, (2) timely response, (3) efficient resource utilization, (4) adaptive resource management, and (5) differentiated services. The goal of the Neptune project is to provide programming and run-time environment support for effective management of services through partitioning, replication, and aggregation. Instead of using monolithic metrics such as throughput, mean response time, etc., the authors define "quality-aware service yield" with respect to a request as denoting the amount of economic benefit resulting from servicing this request in a timely fashion, and then try to maximize the aggregate service yield over all requests.

Service differentiation is done based on service classes, where service accesses of a particular service class obtain the same level of service support. A service class can be a set of client identities, service types, or data partitions. In Neptune, two-level request distribution and scheduling is done: gateways do random polling of the servers and try to achieve load balancing, and service differentiation is done at the servers. This two-level architecture provides scalability and robustness at the cost of less isolation and fairness. Within a server, a request scheduler schedules requests from the queues belonging to different classes to several worker threads such that the aggregate yield is maximized. The offline optimal scheduling problem is NP-complete, and, hence, the authors use heuristics such as Earliest Deadline First (EDF), Yield-Inflated Deadline (YID), Greedy, and Adaptive techniques. The experimental results show that Adaptive outperforms all other heuristics on a 16-node cluster.

RESOURCE OVERBOOKING AND APPLICATION PROFILING IN SHARED HOSTING PLATFORMS

Bhuvan Urgaonkar, Prashant Shenoy, University of Massachusetts; Timothy Roscoe, Intel

The goal of this work is to maximize the number of hosted applications on a server cluster while providing resource guarantees to the applications. Taking a worst-case load and assigning those amounts of resources is not efficient, since the average load of an application is typically an order of magnitude less than the worst case. So the authors propose to use the scheme of overbooking resources and show that this scheme is feasible and maximizes the revenue generated by the available resources.

The authors define "capsules" as the components of an application that runs on a node. To determine the resource requirements of a capsule, the authors perform "application profiling" using Linux Trace Toolkit (for CPU and memory requirements) with well known traces. From typical application profiles, the authors conclude that these capsules exhibit different degrees of burstiness and use "Token Buckets" to represent the resource requirements. A Token Bucket of a capsule with two parameters s and p states that the resource usage of that capsule over any time period t has to be $<= s^*t + p$. Each capsule specifies an overbooking tolerance parameter, O, to denote the probability with which the resource requirements of that capsule can be violated. Once capsules' resource requirements are estimated, these are mapped to nodes using a simple algorithm that uses a greedy technique. A capsule can be mapped to a node only if the resource requirements of the capsule can be satisfied by the node. The experimental results show that there is a 100% improvement with just 1% overbooking.

AN INTEGRATED EXPERIMENTAL ENVIRONMENT FOR DISTRIBUTED SYSTEMS AND NETWORKS

Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar, University of Utah

Typically, network experiments are done through simulation, emulation, or on live networks. While simulation is repeatable but not accurate, live network experimentation is realistic but not repeatable. The emulation method of experimentation is a hybrid approach that creates a synthetic network environment but requires tedious manual configuration. Netbed complements existing experimental environments by spanning simulation, emulation, and live experimentation, integrating them into a common framework. The integration allows ease of use while being realistic. About 2176 experiments were done on the Netbed within the last 12 months by about 365 users.

The Netbed uses a virtual machine approach for network experimentation. Configuration time is improved through automation by two orders of magnitude. Network nodes are emulated using virtual machines on a cluster of nodes. Links including WAN links are emulated using VLANs and tunnels. The network topology to be emulated can be specified either using a ns-type Tcl-based specification or in a Java-based GUI. A global resource allocator scheme assigns local cluster resources to different components of the network topology requested.

Configuring a six-node dumbbell network took just 3 minutes on Netbed, in comparison to a 3.5-hour effort by a student with significant Linux system administration experience.

For more information, see *http://www.netbed.org*.

PEER-TO-PEER INFRASTRUCTURE

Summarized by Scott Banachowski

Scalability and Accuracy in a Large-Scale Network Emulator

Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, David Becker, Duke University

Yocum discussed a network traffic emulator designed to provide realistic scenarios for complex systems such as the Internet. Using the emulator, called ModelNet, has advantages over simulation because it allows execution of real code while still providing control over network conditions not possible with live deployment. The goals when developing ModelNet included support for 10K nodes with a 10Gbps bisection bandwidth, and realistic emulation of network failures and cross-traffic.

The emulator organizes networks into two types of nodes: (1) edge nodes that run the code being tested and connect through (2) core nodes that run ModelNet emulation code. A technique called "distillation" is the key for providing the scalability necessary for handling large numbers of nodes. Distillation transforms the topology of core nodes, which represent the Internet, into a smaller subset of nodes that preserve only interesting links, including the first and last hops of the edge nodes. In this approach, instead of injecting packets that incur processing overhead for an emulator, cross-traffic is simulated by changing the characteristics of the connections through the core nodes.

The ModelNet emulator was verified by reproducing experiments from a previously published study of the CFS storage system layered on the Chord distributed hashtable. Running Chord/CFS on the edge nodes and substituting ModelNet for the network, the throughput of data transfers closely matched the previously published results. Yocum concluded with the assertion that ModelNet is effective for studying how your code behaves in a large-scale network running on its native OS. Questions from audience members revealed that it was not known yet exactly how far ModelNet scales, and that it does require a lot of storage.

More information is available at *http://issg.cs.duke.edu/modelnet.html*.

Pastiche: Making Backup Cheap and Easy

Landon P. Cox, Christopher D. Murray, Brian B. Noble, University of Michigan

Users rarely, if ever, make backups of their personal systems, because it is expensive and time-consuming. Capitalizing on the trend that many disks are often less than half-full, Pastiche is a system for peer-to-peer backups of files on others' computers. Recognizing that many of the binaries on a disk are identical to the binaries of other users, much of the cost of transferring data is eliminated. The goal of Pastiche is efficient, cost-effective backup, while preserving individual privacy.

As its name implies, Pastiche is assembled from already existing technologies. Pastiche uses content-based indexing of data, the same techniques employed by LBFS. Data is fingerprinted and divided into chunks, and a hash function uniquely identifies each chunk. Using only a subset of fingerprints from a disk – for example, a fingerprint from a Windows distribution – Pastiche can identify redundant copies of the data on other machines. To locate machines for backing up data, or "backup buddies," Pastiche uses two overlay networks determined by Pastry, a peer-to-peer routing infrastructure. A mechanism called "lighthouse sweep" was added to Pastry to ensure a geographically diverse set of nodes.

When participating in Pastiche, your system may contain information that also backs up your peers' systems, so the file system must ensure that this data is not deleted or modified. The Chunkstore file system views all data as chunks and assembles files for users in objects called "container files." When data from a container is modified, it is written to a new chunk, preserving the older versions of the data. The performance of backup and restore operations is comparable to VFS copies.

The talk generated enough controversy that there were long lines at the questioning microphones, mostly people interested in more in-depth comparisons with other backup methods.

Secure Routing for Structured Peer-to-Peer Overlay Networks

Miguel Castro, Microsoft Research; Peter Druschel, Rice University; Ayalvadi Ganesh, Antony Rowstrom, Microsoft Research; Dan S. Wallach, Rice University

While peer-to-peer overlay networks are scalable, self-organizing, and robust with respect to node failure, they are susceptible to malicious participants. The talk presented several attacks on these overlays followed by a discussion of defenses.

Castro began with an overview of the Pastry routing overlay and then described several attacks on this technique. In one type of attack, a node can choose its node ID so that, instead of being random, it is positioned to control another node's network access or prevent availability of objects. A defense against this attack would be to certify node IDs using keys from a trusted source. To prevent users from obtaining a large number of node IDs, certificates, it was suggested, might require purchasing. Other attacks on overlays affect routing: for example, supplying peers with fake proximity information or bad routing table information to increase the probability that messages travel through a malicious node. A defense for attacks on routing is to maintain a fallback table, with constrained and more verifiable routing for use when the performance-based routing table fails. Finally, a malicious node may drop or misroute messages. A solution is to incorporate a routing test, and if it fails, rely on a redundant route.

Using these security techniques, peer-to-peer protocols may still work even when up to a quarter of the nodes of an overlay network are malicious, and they provide efficiency when the actual number of compromised nodes is small. In the question period, one audience member quipped that the idea of charging for certificates was the work of the presenter's employer and suggested that the alternative of using a real-world authentication based on a user's identity is more viable.

WORK-IN-PROGRESS REPORTS

Summarized by Scott Banachowski

DISCOVERING BOTTLENECKS IN DISTRIBUTED SYSTEMS

Athicha Muthitacharoen, MIT; Jeffrey C. Mogul, Janet L. Wiener, HP Labs

Contact: Athicha Muthitacharoen, *athicha@amsterdam.lcs.mit.edu*

In large, distributed systems it is not always possible to investigate causes of performance bottlenecks created by internal, proprietary components, because discovering problems often requires instrumenting these components to measure statistics. MIT is developing a tool to identify critical paths using a passive trace of messages. Using the relationships between messages, the tool automatically infers the source of bottlenecks.

WITNESS: LEADER ELECTION WITHOUT MAJORITY

Haifeng Yu and Amin Vahdat, Duke University

Contact: Haifeng Yu, *yhf@cs.duke.edu*

The title must have been inspired by the last presidential election. Many distributed algorithms require a node be elected as leader, but under some kinds of failures it is impossible to guarantee that the elected leader is unique. The new election algorithm provides probabilistic guarantees of a unique leader, and is based on choosing a random set of witnesses to participate in the protocol.

CONFIDENTIAL BYZANTINE FAULT-TOLERANCE

Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, Mike Dahlin, University of Texas, Austin

Contact: Arun Venkataramani, *arun@cs.utexas.edu*

As replication systems add more servers and heterogeneity, they become increasingly vulnerable to attack, so providing confidentiality for replicated data is a difficult problem. This system increases the intrusion-tolerance of a set of replication servers when a number of the servers fail.

INCREASING FILE SYSTEM BURSTINESS FOR ENERGY EFFICIENCY

Athanasios E. Papathanasiou, Michael L. Scott, University of Rochester

Contact: Athanasios Papathanasiou, *papathan@cs.rochester.edu*

This report describes a method to create longer idle times in disk traffic so that these idle periods may be exploited for power saving. The key is to increase the burstiness of access using aggressive prefetching combined with new disk-scheduling algorithms. Trace experiments show the energy reduction using this technique during an MP3 playback reached 55%.

FAB: FEDERATED ARRAY OF BRICKS

Yasushi Saito, Svend Frolund, Arif Merchant, Susan Spence, Alastair Veitch, HP Labs

Contact: Yasushi Saito, *ysaito@hpl.hp.com*

The talk described a logical disk system that uses low-cost commodity CPU and disks and is intended to replace high-end disk arrays. The decentralized system software, based on Petal, achieves high-performance and fail-over ability by replicating disk blocks throughout the cluster.

NCRYPTFS: A SECURE AND CONVENIENT CRYPTOGRAPHIC FILE SYSTEM

Charles P. Wright, Michael C. Martino, and Erez Zadok, Stony Brook University

Contact: Charles P Wright, *cwright@ic.sunysb.edu*

NCryptfs is a stackable file system based on CryptFS from FiST. The low-level file system is transparent to applications. An attach maps an accessed directory to its associated encrypted directory (which stores the actual data in cipher form). Each attach keeps its own data and authorizations private, and on-exit callbacks purge the clear-text data from the kernel.

SUPPORTING MASSIVELY MULTIPLAYER GAMES WITH PEER-TO-PEER SYSTEMS

Wei Xu and Honghui Lu, University of Pennsylvania

Contact: Honghui Lu, *hhl@cis.upenn.edu*

A massively multiplayer game supports up to 200,000 players. Traditionally, games use a client-server architecture, but Wei Xu proposes using peer-to-peer protocols. The talk describes a mapping of players to subsets of multicast groups. By trading consistency for performance, only "nearby" players need to synchronize their environments using P2P multicast groups. A prototype game was developed using Scribe.

KELIPS: A FAT BUT FAST DHT

Indranil Gupta, Prakash Linga, Dr. Kenneth Birman, Dr. Al Demers, Dr. Robbert Van Renesse, Cornell University

Contact: Indranil Gupta, *gupta@cs.cornell.edu*

Kelips is a peer-to-peer probabilistic protocol for group discovery, in which the lookup cost of a file is reduced by enabling the address of any file to be discovered within a single hop. This is achieved by increasing the size of file index tables on each peer and using background communication, or "gossiping," between nodes to keep state updated.

Improvised Network: Autonomously Reconfigurable Mobile Network

Nobuhiko Nishio, Keio University, Japan

Contact: Nobuhiko Nishio, *vino@sfc.keio.ac.jp*

New applications are emerging that use a combination of wireless networks and distributed sensor nodes, as in cellular phones. In such an ad hoc network, both sensors and sink nodes may be mobile, so the research is developing ways to adapt to the changing environment without hurting performance.

Probabilistic Energy Saving in Sensor Networks

Santashil PalChaudhuri and David B. Johnson, Rice University

Contact: Santashil PalChaudhuri, *santa@cs.rice.edu*

On mobile devices, idle and receive periods use about the same amount of energy, so if idle periods may be replaced with inactivity, the device stands to save a lot of energy. According to the "birthday paradox," a relatively small number of people can ensure a high probability that two of them share the same birthday. Applying this principle to communication, only a small number of nodes is needed to ensure that a sender and receiver are active simultaneously. Using a probabilistic-based protocol, the device pre-chooses its waking and sleeping periods, introducing some increase in communication latency but drastically reducing power consumption.

Solar: Supporting Context-Aware Mobile Applications

Guanling Chen and David Kotz, Dartmouth University

Contact: Guanling Chen, *glchen@cs.dartmouth.edu*

The goal of this research is to provide flexible and scalable pervasive computing. Solar is an infrastructure for context computation. An example is a mobile device that subscribes to a set of interesting events; in Solar, by moving the processing of these events to the infrastructure (called "planets"), applications that subscribe to the events remain lightweight. Sharing the computation among several applications reduces both development and network costs.

The exNode Distribution Network

Jeremy Millar, University of Tennessee

Contact: Jeremy Millar, *millar@cs.utk.edu*

exNode is a content distribution network. It is developed to provide access to time-limited data, such as the release of a software product, and is currently used by RedHat. The exNode architecture is effective at distributing load by implementing a highly distributed wide-area RAID system.

Scalable Constrained Routing in Overlay Networks

Xiaohui Gu and Klara Nahrstedt, University of Illinois, Urbana-Champaign

Contact: Xiaohui Gu, *xgu@cs.uiuc.edu*

This system is a step toward value-added service overlays. In overlay networks, such as those used by peer-to-peer applications, it is desirable to satisfy some end-to-end constraints – for example, establishing a level of quality of service between endpoints. Qualay is a proposed overlay network designed to provide QoS constraints over paths. In the setup phase, service paths are chosen by probing nodes, and in the runtime phase, faults are detected and paths rerouted to maintain QoS.

Reverse Firewalls in Denali

Marianne Shaw and Steve Gribble, University of Washington

Contact: Marianne Shaw, *mar@cs.washington.edu*

Shaw presented a way to introduce policies and mechanisms to protect the Internet from bad services. The system allows untrusted code to run in the network infrastructure on a virtual machine, with a reverse firewall that prevents the Internet from malicious traffic generated by the VM. The flexible framework allows policies to be added on the fly, and in the example provided in the talk, Shaw focused on a "don't speak unless spoken to" policy for containment of client-server code.

Improving Application Performance Through System Call Composition

Amit Purohit, Joseph Spadavecchia, Charles Wright, Erez Zadok, Stony Brook University

Contact: Amit Purohit, *purohit@cs.sunysb.edu*

A problem with application performance is overhead incurred by system calls that move data across the kernel boundary. This system provides a solution that removes user-level bottlenecks by moving user code into the kernel. Using a tool called Cosy, combined with the gcc compiler, designated code is compiled into special code segments that can be loaded into the kernel at runtime. Static and dynamic checks ensure that kernel security is not violated, and adding preemption to the kernel protects against user segments monopolizing the CPU.

Performance of Mach-Kernel

Igor Shmukler, OS Research

Contact: Igor Shmukler, *shmukler@mail.ru*

Shmukler spoke about enhancements to the Mach kernel aimed at increasing its attractiveness to the user community. Although Mach introduced many good ideas, it didn't really catch on because it was never fine-tuned for the common-case performance. Shmukler tried to clear Mach's bad name by discussing proposed improvements, including changing the memory management subsystem, optimizing the RPC implementation, adding new synchronization primitives, and stomping on a slew of bugs.

Elastic Quotas

Ozgur Can Leonard, Jason Nieh, Erez Zadok, Jeffrey Osborn, Ariye Shater, Charles P. Wright, Kiran-Kumar Muniswamy-Reddy, Stony Brook University

Contact: Jeffrey R. Osborn, *jrosborn@ic.sunysb.edu*

"Elastic quotas" for disks are aimed at shared file servers, such as those used by university students, where each user receives a quota of space. By implementing elastic quotas, extra space may be allocated to users for their temporary use, but this space may later be reclaimed. The elastic quota service sets both global and user-assigned policies for how the space occupied by files designated as elastic will be reclaimed, using information such as size or creation time. The next step in their research is to determine if users will embrace such a system.

A Mail Service on OceanStore

Steven Czerwinski, Anthony Joseph, John Kubiatowicz, University of California, Berkeley

Contact: Steven Czerwinski, *czerwin@eecs.berkeley.edu*

The Mail Service uses the OceanStore file system to provide low-latency access to email, independent of a user's location. Goals of the system include data durability and relaxed consistency by allowing application-specific conflict resolution. Following this session, members of the project gave a demo of OceanStore.

NETWORK BEHAVIOR

Summarized by Kenneth Yocum

An Analysis of Internet Content Delivery Systems

Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, Henry M. Levy, University of Washington

There's a lot more than just Web content being served across the Internet. Now we have CDNs and peer-to-peer systems serving up audio, video clips, and movies. The authors studied HTTP Web traffic, Akamai CDN, and Kazaa and Gnutella nets. The basic result of the authors' trace, conducted at the University of Washington, is that peer-to-peer traffic constitutes a large fraction of the bytes, and it's very different from the Web. For example, it may be possible to cache 80–90% of the outbound traffic and 60% of the inbound traffic. But it takes a long time to warm up the cache (about a month). In both directions P2P objects are three orders of magnitude larger than Web objects. A small number of objects account for most of the bytes in P2P systems.

TCP Nice: A Mechanism for Background Transfers

Arun Venkataramani, Ravi Kokku, Mike Dahlin, University of Texas, Austin

TCP NICE, a building block for background transfers, finds and uses spare bandwidth in the Internet to improve availability, reliability, latency, and consistency. As a "new" variant on TCP congestion control, TCP NICE is similar to TCP VEGAS monitor RTT but provides three changes: a more sensitive congestion detector, multiplicative reduction in response to increasing RTT, and the possibility of having a congestion window less than one. With NICE you can bound the interference caused by background flows. One use is prefetching. The authors found that NICE could improve performance by a factor of three, where using old-style TCP hurt performance by a factor of six.

The Effectiveness of Request Redirection on CDN Robustness

Limin Wang, Vivek Pai, Larry Peterson, Princeton University

We now use replication across geographic distance to deliver content. Client requests are delivered to the "best" candidate based on server load, server closeness, and cache. This work describes current schemes and introduces a new one to balance locality, load, and nearness (proximity). This new scheme was shown through simulation to improve system capacity by 60–90% while maintaining low request latencies for clients. One dynamic algorithm, Fine Dynamic Replication (FDR), is especially promising. It keeps fine-grained information on URL popularity to adjust the number of replicas. They're trying to deploy it on PlanetLab.

MIGRATION

Summarized by Richard S. Cox

The Design and Implementation of Zap: A System for Migrating Computing Environments

Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh, Columbia University

Zap supports the transparent migration of unmodified applications. The migration of network applications is supported without loss of connectivity. Zap-migrated processes leave no residual state behind on the previous system. Implementing Zap involves minimal changes to a commodity operating system and requires low overhead.

Three problems must be solved to migrate processes: resource consistency, resource conflicts, and resource dependency. Zap's solution to all three is the process domain (pod). A pod is a private virtual space that may contain a single process, a process group, or a whole user session. As a private space, processes in a pod cannot interact with processes outside a pod. Pods are migrated as a unit. Zap contains pods by introducing a thin layer in the Linux kernel, virtualizing process IDs, IPC, memory, the file system, network, and devices. The overhead of this approach is minimal, and the pod images are small.

More information can be found at *http://www.ncl.cs.columbia.edu/research/migrate*.

Optimizing the Migration of Virtual Computers

Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, Mendel Rosenblum, Stanford University

By virtualizing the x86 architecture, the VMware GSX server enables an entire virtual machine's (VM) hardware state to be easily suspended and captured. Once saved, the state can be sent to another machine and resumed. However, capturing the entire state generates machine images, or capsules, that are gigabytes in size. This work applies several optimizations to reduce the capsules to a size that can be transferred over a DSL link in under 20 minutes, enabling applications such as user mobility and software updates. The two largest components of a capsule are the disk and memory images.

Using standard copy-on-write techniques, VMware can track the changes to a disk image and transfer only the differences if the target machine already has an old version of the disk image. By hashing each disk block, and searching for a block with matching hash value on the target system, the server can avoid transferring pages whose contents already exist on the target system. Much of a VM's memory may not be in active use; thus, if VMware could request that the guest OS de-allocate inactive pages, the size of the memory image could be greatly reduced. This is the idea behind ballooning, which utilizes a driver added to the guest OS to reclaim low-priority pages prior to suspending the VM. Finally, by demand-paging the disk images, the time to resume the VM on the target can be reduced. Demand-paging takes advantage of the disk-latency tolerance already built into modern OSes. Several macro-benchmarks show that the combination of these techniques is effective in reducing the total data transferred to migrate a capsule as well as the time-to-start.

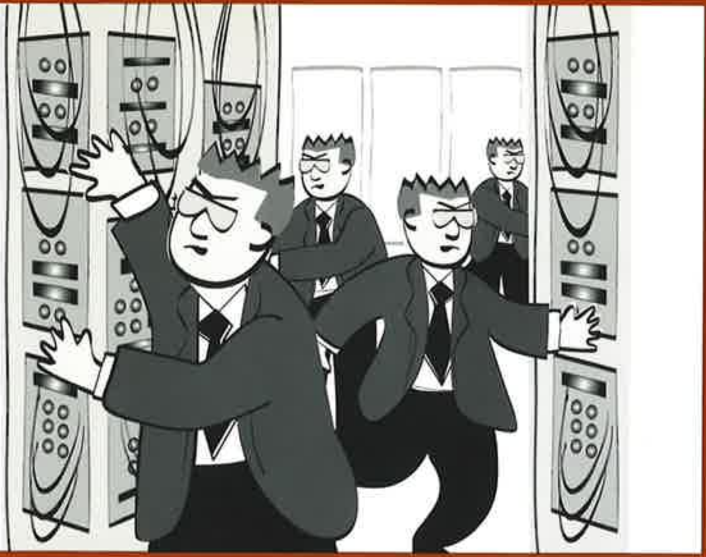
Luna: A Flexible Java Protection System

Chris Hawblitzel, Dartmouth College; Thorsten von Eicken, Expertcity

Extensible applications require protection schemes that can isolate extensions while permitting lightweight communication. Java uses language-based approaches to enforce domain separation, enabling cheap communication because of the single address space. However, systems with Java extensions lack clear domain boundaries; all code and objects are stuck together. The resources used by an extension cannot be reclaimed if the extension is terminated, because they may be referenced by other parts of the system.

By introducing a task abstraction, extensions in a Java system can be strongly isolated. Tasks contain all the objects, threads, and code for an extension. All cross-task communication is explicit. In Luna, regular (local) pointers are not allowed to reference objects in other tasks. Remote pointers, a new type of reference that is allowed to point to objects in other tasks, are Luna's mechanism supporting intertask communication. Remote pointers may be revoked at any time; if a revoked remote pointer is used, an exception is raised. This allows an entire extension to be removed from the system cleanly, without dangling references in other tasks. Remote pointers are implemented with a two-word structure. The first word is just the memory address of the object. The second word is a pointer to the permit, which contains a revocation flag and is checked before each use. As an optimization that removes most checks in common cases, Luna can generate loop code that does not contain any checks. On revocation, threads using the object are suspended, and a breakpoint is placed where the check would have been. If and when the breakpoint is reached, an exception is raised, simulating the effect of the check. Micro-benchmarks, as well as an implementation of an extensible Squid Webcache, confirm that Luna's isolation imposes low overhead.

Which monitoring solution is right for you?



(a) Agents

(b) Agentless

Before you purchase a systems and Web monitoring solution, take this quick multiple-choice test:

1. Do you prefer (a) complications, or (b) efficiency?
2. Would you rather install software on (a) every server you want to monitor, or (b) one server?
3. Would you rather spend your time maintaining software on (a) many servers, or (b) a single monitoring server?
4. Is your priority (a) installing the biggest solution, or (b) maximizing ROI?

If you answered (b) to these questions, SiteScope is the monitoring solution for you.

Try SiteScope today.
Download a FREE 10-day trial at:
www.mercuryinteractive.com/foryou

MERCURY INTERACTIVE

2003 USENIX

SPONSORED BY USENIX, THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Annual Technical Conference

JUNE 9–14, MARRIOTT RIVERCENTER, SAN ANTONIO, TEXAS

Three days of tutorials
Three days of technical sessions
Two days of vendor exhibits
Unlimited learning opportunities

Register by May 16 & Save Up to $300
http://www.usenix.org/usenix03/

;login:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710