

Demystifying Sendmail



*Hal Pomeranz
Deer Run Associates*

Demystifying Sendmail

All material in this course Copyright © Hal Pomeranz and Deer Run Associates, 2005-2006. All rights reserved.

Hal Pomeranz * Founder/CEO * hal@deer-run.com
Deer Run Associates * PO Box 50638 * Eugene, OR 97405
+1 541-683-8680 (voice) * +1 541-683-8681 (fax)
<http://www.deer-run.com/>

Demystifying Sendmail.....	1
Welcome!	5
About This Course	8
Course Organization.....	10
An Important Message from Your Instructor.....	12
Sendmail Basics	13
Sendmail as an Email “Backbone”.....	14
The Argument for Multiple Layers	18
What is SMTP?	21
MTA vs. MSP	23
How Does This Really Work?	29
Open Source vs. Vendor Provided Sendmail.....	35
Creating Sendmail Configuration Files.....	37
Exercises.....	39
Answers to Exercises	44
Internal Relays.....	50
The <code>mailertable</code> Feature	54
Other Configuration Directives.....	59
Masquerading.....	61
Exercises.....	64
Answers to Exercises	67
External Mail Relays.....	70
Operating System Security.....	71
External Sendmail Posture	75
No Promiscuous Relays!	76
Simple External Relay Configuration	79
Additional Configuration	83
Limiting Sendmail Privileges.....	86
Other Considerations.....	91
MAIL_HUB Oddity	93
Exercises.....	95
Answers to Exercises	101
Sendmail Administration.....	108
What Version of Sendmail is Installed?.....	109
The Sendmail Queue	111
Stopping and Starting Sendmail.....	119
Testing Sendmail.....	125
Exercises.....	137
Answers to Exercises.....	141

Performance Tuning.....	148
Don't Tune If You Don't Have To	149
Increasing Throughput Through Parallelization	150
Typical Sendmail Performance Bottlenecks	152
Synchronous Writes	153
Problems with "Deep" Queues.....	155
DNS Issues	159
Tuning Timeout Values.....	161
Throttling Controls.....	163
Exercises.....	165
Answers to Exercises	167
Spam and Virus Control.....	170
Before You Begin.....	171
Sendmail's Built-In Anti-Spam Features.....	177
The Sendmail Milter Interface	186
Now I'm Really Confused.....	194
Exercises.....	195
Answers to Exercises	197
Virtual Domains	200
What Are We Talking About?	201
"Parallel" Domains.....	202
Virtual Domains	204
Deployment Issues	205
The <code>virtusertable</code> Feature	208
Other Foreign Domains.....	212
Exercises.....	213
Answers to Exercises	216
Aliases and Mailing Lists.....	221
Aliases Overview	222
Some Sample Aliases Entries.....	224
When Alias Expansion Happens.....	226
Setting Up a Machine to Handle Aliases	228
Dealing With Common Issues.....	230
About Mailing Lists	236
Exercises.....	239
Answers to Exercises	245
Wrap Up.....	252
Any Final Questions?.....	253
Things I Need to Mention Before I Go	254
Books and URLs for Further Reading.....	256
Thanks!.....	259

Appendix A: Majordomo and Mailing Lists	260
About Majordomo	261
Setting Up Majordomo	262
Creating a Mailing List	266
Appendix B: Writing Email Auto-Responders	270
Why Should I Learn This?	271
High-Level Overview	272
Our Example	273
The “Envelope From”	276
The Rest of the Headers	277
Invoking Sendmail Very Carefully	279
Creating Our Outgoing Email Message	280
Finishing Up	282



Who is Hal Pomeranz?

- Independent IT Consultant
- Course author and senior instructor for the SANS Institute ("the Unix guy")
- Technical contributor to Unix guidelines from the Center for Internet Security
- Tech editor for *Sys Admin Magazine*

This is usually enough to keep me busy...

Welcome!

My name is Hal Pomeranz and I'm an independent IT consultant with almost 20 years of experience in Unix systems, networking, and computer security. While I'm probably better known for my computer security work, I've always loved hacking around with Sendmail (and related subjects like DNS, mailing lists, etc) and try to "keep my hand in" by taking Sendmail-related contracts whenever they come up. Hence this course.

I also have a number of other hats I wear in addition to my consulting business. Some of you may know me due to my association with the SANS Institute (www.sans.org), probably the top provider of computer security training in the US today. I have the curious distinction of being SANS' "oldest" instructor—in terms of longevity with the organization, not age—having taught my first course for SANS back in 1994 (and I actually presented at earlier SANS conferences). Basically, I'm SANS' "Unix Security Dude": I'm the primary author of their six-day Unix Security Training as well as the author of the Unix material in their introductory "Security Essentials" course. I even used to teach a DNS and Sendmail course for SANS, although we haven't offered it in quite a while (you can find the last version of that course in PDF form on my web site—<http://www.deer-run.com/~hal/dns-sendmail/>). You'll also find a number of other articles on Sendmail, DNS, and related subjects on my web site.

I'm also one of the Unix security dudes for The Center for Internet Security (www.CISecurity.org). CIS publishes consensus security guidelines for various operating systems—not just Unix systems, but also Windows, Cisco IOS, etc—and also applications like Oracle, IIS, etc. I helped to develop the original format for the Unix documents published by the Center and am the maintainer for their Solaris guidelines.

And since you can't have enough Unix in your life, I also serve as the Technical Editor for *Sys Admin Magazine*, published by CMP Media. The nice thing about being the Tech Editor is that it gives me an excuse to sit down and read all of the articles in every issue we publish—plus I get to read them about 3 months before everybody else does! If you administer Unix systems, I think you'll find *Sys Admin* a useful publication, and if you don't find it useful then we'd like to hear from you.

Normally all of this keeps me quite busy, but I like to remain active in the Unix/Linux and System Administration communities. At various times I've served on the Board of Directors for BayLISA (SF Bay Area), BBLISA (Boston), and USENIX. I also regularly give talks for local user groups wherever my travels take me. In 2001 I was awarded the "Outstanding Achievement Award" from the SAGE system administration special interest group of USENIX, which is a really neat award since it comes from your peers in the community.



What does *he* know about Sendmail?

- First given root in 1987-- specifically so I could get Sendmail working!
- Since then I've managed email for lots of different organizations, including:
 - 15 months managing the external corporate email gateways for Cisco
 - Six months rebuilding the corporate email system for Microsoft WebTV
- These days I also support email for the most demanding user of all-- *my wife!*

As you've probably gathered by now, I've spent a lot of my life working with Unix systems. Because Sendmail is the default mail system for Unix, you might guess that I have at least passing familiarity with the intricacies of Sendmail. In fact, my first exposure to Unix administration was due almost entirely to Sendmail. As a junior system administrator, I was given all the jobs that nobody wanted—in particular my first real assignment as a sys admin was to figure out Sendmail and get email working for the site I was supporting at the time.

When you get a reputation as “the Sendmail guy” it tends to stick with you, and it seems like I ended up working with Sendmail at most of the jobs I had from there on out. Before I became a consultant, I set up and ran the Internet email systems for QMS (the old laser printer company) and NetMarket (an early Internet start-up). While consulting at Cisco, I ended up running their external corporate email gateways—I was supposed to be there helping the corporate information security team, but we ended up “owning” the external email gateways through an accident of politics and the rest is history. I also had an interesting contract rebuilding the internal corporate DNS and email infrastructure at Microsoft's WebTV division (now MSNTV) after a previous outsourced IT group had left it in a shambles. Despite being a division of Microsoft, WebTV at the time was primarily a Unix shop.

I also currently provide email services and support for a number of small companies, as well as for friends and family (one of the hazards of being the lone computer geek in a family). Of course I also have to manage email for my own business, which I run with the help of my wife Laura. And believe me, when email isn't working for Laura or when we start getting too much spam, she makes sure that fixing things is my *TOP* priority!



What Is/Is Not This Course

- This course is:
 - Designed to get Sendmail newbies to a point where they are self-supporting
 - Primarily concerned with Sendmail running as a mail "relay" rather than a mail "depot"
- What this course is not:
 - Complete coverage of the subject
 - Free of religious belief
 - The "Hackers Guide" to Sendmail

About This Course

The way we use Sendmail has certainly changed a lot in the nearly 20 years I've been working with it. When I first got into the business, Unix systems made up the vast majority of systems used to transport email across the Internet and store email for users. Sendmail was practically "the only game in town" when it came to email systems.

Since that time, we've seen two major changes. The first is the rise of Windows-based email servers—primarily Microsoft Exchange and Lotus Domino these days. Very few organizations store user email on Unix systems anymore, which has changed the ways in which Sendmail is used these days—or perhaps "limited Sendmail's scope" is a better description. The other change has been the development of other Unix mail server software, including QMail, Postfix, and Exim. While Sendmail is still the default mail system for all Unix-like operating systems, the development of competing mail systems in the Unix space has helped shape the development of Sendmail and there has been quite a bit of "cross-pollination" of features and design ideas between different email software.

This course tries to focus on just the material people need to know to use Sendmail in its current "standard" deployment—primarily as a series of relays to get email from a set of Windows-based email servers at one organization to the email servers at other outside organizations across the Internet. We will also look at the problem of using Sendmail as an "email router" within a large, decentralized organization as well.

Sendmail is an incredibly complicated program that is almost infinitely configurable. There is no way to cover everything about Sendmail in a one or two day course—the standard Sendmail reference book, O'Reilly and Associates' *Sendmail*, is 1200 pages long by itself (if you include the index). So the goal here is to first provide students with

actual working examples that they can use immediately to route email around and in and out of their organization, but also to introduce enough of the concepts and terminology used by Sendmail administrators so that students will be able to use books like *Sendmail* when they need to solve a problem or build an architecture that's not covered by this course.

Since we can't cover everything about Sendmail in the time allotted, I'm necessarily going to be telling you only what I think is "important" for you to know. There are lots of other people with Sendmail expertise out there and even other Sendmail courses that you can choose from. So my opinion of what's important may not agree with everybody else's. I also have strong opinions in several areas about the "right" way to configure Sendmail for various tasks, although others may disagree. Wherever possible, I'll try and point out these areas of "religious disagreement" and let you make up your own mind.

Many Sendmail courses out there cover the complex internal ruleset language used by Sendmail in its configuration files. My belief is that people who are new to Sendmail administration don't need to get bogged down in this level of detail. Everything you need to do to configure Sendmail to handle even fairly complicated email routing and delivery can be done via the higher-level "macro configuration" interface that we will be exploiting throughout this course. The goal of this first course in Sendmail is not to make you "Sendmail hackers" right out of the gate. The goal is to get you up and running with Sendmail and being a useful email administrator as fast as possible. Trust me, once you master the basics you'll end up sliding into the "Sendmail hacker" role gradually over time.

This course is based primarily on Sendmail v8.12.x and later. Sendmail v8.12 marked a fairly significant architectural change in the way Sendmail operates, and it's worth your while to upgrade if you're still running an older release. While much of the material in this course applies to older releases, specific syntax in this course may cause problems if you try to use the examples as written on older versions (consult the O'Reilly *Sendmail* book if you're using older releases). This course does not cover any material related to the "next generation" Sendmail release, Sendmail X, which is still being developed.

the platform it's running on, as well as basic DNS configuration needed for email routing.

- *Sendmail Administration* – Knowing how to configure Sendmail is only the first step. As an email administrator you also have to be able to keep an eye on how Sendmail is doing. Are the mail queues backing up? How do you read Sendmail's logs to diagnose both normal and abnormal behavior? How can you test to make sure your current email configuration is working, and how can you test new configurations before deploying them into production and possibly breaking your existing setup?
- *Performance Tuning* – On modern hardware, Sendmail actually performs extremely efficiently in its default configuration. However, it's worthwhile to discuss some of the common bottlenecks you run into on high-volume mail relays and ways you can address these issues. Frankly, understanding Sendmail's performance characteristics also gives you a better understanding of how Sendmail functions "under the hood".
- *Spam and Virus Control* – Because of its function as an email relay, Sendmail is often called upon to also act as a filter for incoming (and outgoing) email spam and viruses. This section examines Sendmail's built-in functionality for spam control, and introduces you to the so-called Milter ("Mail Filter") interface that allows you to plug in all sorts of third-party spam and virus control tools (both freeware and commercial).
- *Virtual Domains* – Most organizations need to support email for many other email domains besides their primary email domain. This section looks at various pieces of Sendmail functionality that allow a single Sendmail server to pretend to be the email server for many different organizations.
- *Aliases and Mailing Lists* – Sendmail has powerful and flexible functionality for creating aliases for email distribution lists, email archiving, and email auto-responder type programs. I think Sendmail is better at this than Windows-based email systems, but of course I'm a huge "Unix bigot" and have been working with Unix and Sendmail for almost 20 years, so what else would I think? There is also very good mailing list management software available for Unix systems. In this module we'll take a look at all of this technology and also explore how you might configure a Unix system within your organization for doing this.

After the wrap-up material at the end of the course, I've also included some additional information in two Appendices. The first Appendix is material specifically on setting up and managing the older Perl-based Majordomo mailing list software. The "hands-on" exercise after the last module in the course will introduce you to the newer Mailman mailing list software, but lots of people still like to use Majordomo. The other Appendix is information for people who are interested in correctly and securely writing their own custom auto-responder type programs. Not everybody who takes this course is enough of a programmer to care about this material, but I include it here for those of you with the appropriate bent.



Your job...

ASK QUESTIONS!

An Important Message from Your Instructor

I've been working with Sendmail and Unix for a long, long time. Things that I regard as "obvious" and skip over quickly often turn out to be not very obvious to you, my students. So if you're confused by something, need clarification, or simply are tired of hearing me jabber away at you, throw something or raise your hand to get my attention and try and phrase your confusion in the form of a question. Trust me, if you're wondering about something, chances are that most of the rest of the class is wondering about it as well.

The course is paced to allow people to ask lots of questions and it works much better as an interactive conversation between you and me. Sometimes if I think we're headed down a tangent or "rat hole" that isn't interesting to most of the other people in the class, I'll defer your question to one of the break periods where I'll be happy to go into as much detail as you want. Or if you're shy of asking a particular question in front of the class, I'm happy to answer questions for individuals during the break.

My contact information is available on the first page of this course book, and I'm happy to answer questions for former students (and even random people on the Internet, time permitting). Business cards are available on request, if you'd like my contact information in a more portable format.

Sendmail Basics



Sendmail Basics

This section is a high-speed introduction to the Sendmail architecture and terminology we'll be using throughout the rest of this course. After completing this module you should:

- Have a basic understanding of the Sendmail relay architecture used by this course
- Know what the Simple Mail Transfer Protocol (SMTP) is and how it relates to Sendmail
- Understand the distinction between a Mail Transfer Agent (MTA) and a Message Submission Process (MSP) and how Sendmail acts as both
- Be familiar with the basic configuration files and directories associated with Sendmail
- Understand at a high-level how email is created, handed off, and routed through Sendmail on Unix systems
- Understand the basics of how to generate new Sendmail configuration files



Sendmail as a "Backbone"

- Sendmail typically routes email between PC-based corporate email environments
- Often use a "layered" approach:
 - "External Relays" handled spam and virus controls and Internet email routing
 - "Internal Relays" handle special-case internal corporate email routing rules
- Lots of different permutations possible!

Sendmail as an Email "Backbone"

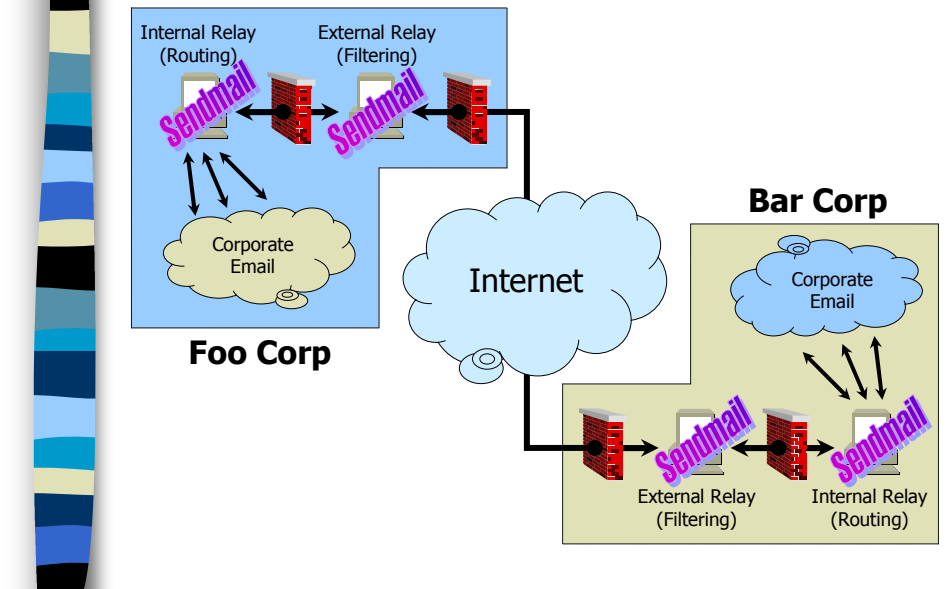
As I mentioned earlier, it is fairly rare these days for organizations to use Unix systems running Sendmail as actual mail "servers" where user mailboxes are stored. Instead, organizations typically use Sendmail merely as email routing "backbone" or system of "relays" that helps move email from one place to another—usually from Exchange or Domino at one company to Exchange or Domino at another company elsewhere on the Internet. Sendmail is also very good at getting email from point A to point B within a single company, if you have a distributed kind of email environment with lots of different groups of email servers scattered about.

When deploying Sendmail as a relay, organizations will often set up two distinct layers of Sendmail relays. The "external relay" layer handles the organization's email communication to and from the Internet. As the initial "choke point" for all email entering the organization, this is typically the place where anti-spam solutions are placed, and often other sorts of filtering as well, such as anti-virus scanners. The "internal relay" layer mostly handles internal routing issues within the organization, hiding this complexity from the outside world. As organizations become more concerned about email virus threats from within, you're starting to see the anti-virus piece of the equation migrating further and further "inwards" towards these internal relays.

However, there are as many ways to deploy Sendmail as there are sites on the Internet. While this separation between "external" relaying and anti-spam filtering vs. "internal" routing is a useful conceptual framework, these layers don't always exist as separate physical entities. Some organizations may only have a single layer that handles both filtering and routing between their internal Windows-based email system and the Internet. Some sites don't run Unix-based mail servers at all and use email "appliance" type

systems to handle filtering and routing. Others simply have their Exchange or Domino servers communicate directly to the Internet, though personally I think this is insane behavior from a computer security perspective.

Simple Sendmail Architecture



The slide above shows pictorially the sort of “external” vs. “internal” relay architecture we’re discussing here, with the positioning of these relays between the multiple layers of network firewalls used by most organizations. On the left we have Foo Corporation that has some collection of Windows-based email servers for their user community, as represented by the “Corporate Email” cloud. When Foo Corp’s users want to send email outside of the company (or possibly to users elsewhere within Foo Corp), the Windows-based email environment forwards that email up to the “internal” relay layer for intelligent handling. These internal relays will make appropriate email routing decisions based on the destination address on the email. In many cases, the email will be destined for some outside organization, so the internal relays must hand that email off to the “external” relays that communicate to and from the Internet. These external relays use standard Internet email routing algorithms (which we’ll discuss shortly) to find the appropriate email server at other organizations to send this email to.

Let’s say the email message is destined for some user over at Bar Corporation, represented here on the righthand side of our picture. Typically, the external servers for Foo Corp will end up communicating with the external servers at Bar Corp to transmit the email. At this point, the external servers at Bar Corp will apply their anti-spam and anti-virus scanners to the incoming email, just to make sure it’s email that the folks at Bar Corp are interested in receiving. Beyond that level, however, the external servers for Bar Corp probably lack the “intelligence” to properly route the incoming email to its final destination within Bar Corp. So the external servers pass the email inward to the internal relays for further routing to the appropriate (Windows-based) message store for the recipient.

As the picture illustrates, the places where Sendmail gets deployed in this picture are those internal and external relay servers. It need not be Sendmail of course—you could use an “appliance” type system or even different Unix-based mail server software—but since this is a course on Sendmail, we’re going to assume you’re planning on using Sendmail for this.



Why Two Layers?

- Different security postures
- Scale at different rates
- Isolate complexity
- Change management
- Organizational boundaries
- Outsourcing

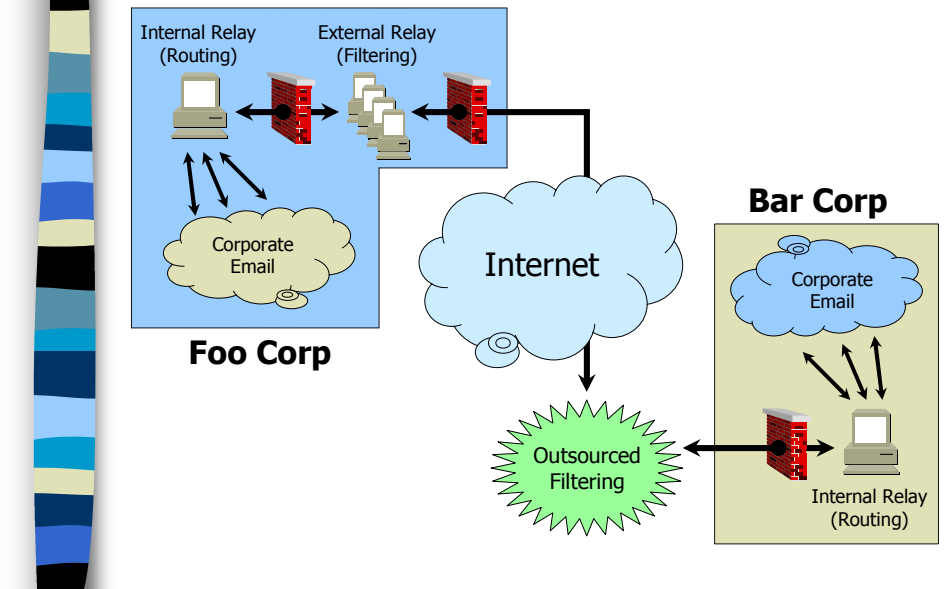
The Argument for Multiple Layers

It may seem cumbersome at first to deploy multiple layers of email relays with all of the firewalls, network routers and switches, etc that go along with them. But there are many reasons why this architecture has tended to evolve the way that it has.

- Probably the most important issue is the differing security needs for the internal vs. external gateways. The external email gateways for an organization are tempting targets for external attackers—critical machines that are well advertised with direct, more or less unfiltered Internet access. As we'll discuss, you want to spend a good deal of time looking to the security of your external gateways. However, complexity and security are usually at odds, so you want your external gateways to be stripped-down, locked-down machines that just have enough intelligence to grab email and throw it into your internal email environment (or the reverse). Your internal gateways, on the other hand will typically need a richer set of functionality for accomplishing their mission, and not being directly connected to the Internet can be operated in a more relaxed security posture.
- What you'll also discover is that the workload of the internal and external relays scales at different rates. As the amount of spam on the Internet continues to increase, and as we need more and more clever solutions for filtering out the spam from the valuable email, you may find yourself needing to split the incoming email load across multiple external relays. This is only more true if you're also doing virus scanning on incoming email, which tends to have a very heavy performance footprint. On the other hand, once all of the spam and virus-laden email is filtered out, you may find that your actual incoming email volume is hardly ramping up at all and only a single internal email relay is required (or perhaps a couple for the sake of redundancy).

- The configuration of an external mail relay that's doing anti-spam and anti-virus work is a peculiar and complicated thing, with lots of third-party programs in addition to Sendmail to handle various filtering tasks. The configuration of your internal relays with all of their special-case routing rules for different parts of your email infrastructure is also a deeply mysterious domain. So it helps to separate these two different types of complexity into separate physical layers to make management easier. That way, you can go into your internal email routing rules and make tweaks without worrying about changing something that breaks your spam filters, and vice-versa. You don't end up having to understand the entire end-to-end email infrastructure to make changes in just one area.
- More generally, having multiple layers makes things easier from the perspective of making any sort of change. Maybe a new Sendmail security hole is discovered—it's vital that you update your Internet-facing mail servers as soon as humanly possible, but maybe you could wait to upgrade your internal servers until you see how the new version works out. Or when you're upgrading the operating system of the platform your mail servers run on, it may be necessary to delay the OS upgrades on your external relays until the vendor of your third-party anti-spam and anti-virus products certify those products for the new OS version. In the meantime, you can be upgrading the OS on your internal mail relays to stay in line with your corporate standards.
- Having things split out into multiple layers also allows you to let different groups manage different aspects of your email architectures. Maybe the external servers are under the control of the group that manages your web servers and other Internet-facing devices while the internal relays are "owned" by the internal corporate IT group.
- Perhaps your company doesn't even want to own some aspects of its email system. As spam and virus control become more and more complicated, a lot of companies are looking at outsourcing this problem to somebody else. Basically in this model the company no longer owns the "external" filtering layer, but can still maintain their own "internal" routing layer for various specialized internal needs. Or maybe the organization purchases a "shrink-wrapped" or "appliance" type solution and plugs that in instead of a self-supported Unix server running Sendmail.

Tweaking the Architecture



As an example, the picture shown here illustrates some of the ideas we just discussed. Foo Corp has found it necessary to deploy multiple machines in parallel to act as external relays for their inbound/outbound email. Bar Corp has decided that's too much trouble and just contracted with an outsourcing provider to handle spam and virus scanning.

So now all of Bar Corp's incoming email goes through their outsourcing vendor's infrastructure first before being relayed into their internal email relays. If Bar Corp is small enough, that internal relay might even go away and the outsourcing provider would just shove the email directly into Bar Corp's Exchange or Domino servers. Or going the other way, Bar Corp might decided to keep both internal and external email servers around to handle outgoing communications with other outside organizations—though usually the outsourcing provider will handle both Bar Corp's incoming and outgoing email.

Anyway, like I said, there are as many ways to deploy this architecture as there are sites on the Internet. But for purposes of our discussions in the rest of the course we will assume that our hypothetical organization has physically separate internal and external relay servers, both running Sendmail on some flavor of Unix.

to make lots of DNS queries to the name server(s) configured in `/etc/resolv.conf`. DNS typically happens on 53/udp (and sometimes 53/tcp).

Software that receives, processes, and routes email via SMTP is referred to as a Mail Transfer Agent (MTA). Sendmail is an MTA, as are QMail, Postfix, Exim, and all of the other Unix SMTP servers. Most Windows-based mail servers are now capable of speaking SMTP natively, or an SMTP gateway product is available for an additional charge.



MTA vs. MSP

- Message Submission Process (MSP) gets new email to MTA for SMTP delivery
- On Unix systems, Sendmail normally acts as both MSP and MTA
- This course is primarily interested in the MTA features of Sendmail
- But in the "normal" case:
 - 99.9% of your Unix systems are *not* MTAs
 - Disable MTA functionality for security

MTA vs. MSP

So an MTA is responsible for receiving and processing email that comes in from outside the system, but what about email that's created on the local machine—whether by local users or by automated processes such as `cron` jobs and the like? The job of the Message Submission Process (MSP) is to take all such locally created email and hand it off to an MTA for processing and SMTP delivery (although the MTA may decide that the right thing to do is deliver the email to a local user mailbox on the same system). In the Unix world, the MSP will typically use SMTP to relay the email to a nearby MTA.

In fact, in the Unix universe Sendmail acts as both an MTA and an MSP. All Unix systems are capable of generating outgoing emails via Sendmail in its role as MSP, but only a very small fraction of the Unix machines in a given organization are actually acting as MTAs. What's odd about this course is that we're going to spend nearly all of our time focusing on Sendmail's role as an MTA, when in reality the advice I'm giving you doesn't apply to the 99.99% of Unix machines in your environment that are functioning as MSPs only.

Even more strange is the fact that nearly every Unix system's default configuration is to have an active MTA running on the machine, listening on port 25/tcp for incoming email that it's never going to receive. In fact, the only thing that MTA process is doing by listening on port 25 is making you vulnerable to the next automated worm that gets written to exploit some as yet undiscovered vulnerability in Sendmail. If you want to drastically improve the Sendmail security posture at your site, disable the MTA configuration of Sendmail on all Unix machines that are not acting as mail servers (more on this as we go along).

The problem with doing this, however, is that the default MSP configuration for Sendmail on most Unix machines is to submit newly created email to the MTA *on the local system* for processing. If you shut off the MTA on the local machine, then outgoing email can't escape. However, while this is the default configuration for the MSP, it's not the only possible solution. You could have the MSP submit the outgoing email to an MTA running on some other machine—even a Windows system with an SMTP server running. In a couple of slides I'll give you some directions on how to make this happen.

I don't want to go too far down a rat hole as far as MSP configuration issues, because that's not the primary focus of this course. If you happen to have a lot of Unix systems in your environment that are not acting as mail servers, I urge you to read a couple of articles I've written for *Sys Admin Magazine* on this subject:

<http://www.deer-run.com/~hal/sysadmin/sendmail.html>

<http://www.deer-run.com/~hal/sysadmin/sendmail2.html>



Sendmail as MTA

- Listen for email and manage queue:
`/usr/sbin/sendmail -bd -q1h`
- Runs as `root` user
- Important files in `/etc/mail`:
 - `sendmail.cf` -- main config file
 - `aliases` -- distribution lists, et al
 - `local-host-names` -- local email doms
 - `relay-domains` -- *more on this later...*
- Queue dir: `/var/spool/mqueue`

Sendmail as MTA

Sendmail uses one set of command line options as well as configuration files and directories in its MTA configuration and a completely different set when operating as an MSP. And normally when you look at the process table of a Unix machine you'll see two different `sendmail` processes running—one is the MTA and the other is related to the MSP.

The MTA process is the one that is started with the following arguments:

```
/usr/sbin/sendmail -bd -q1h
```

The “`-bd`” flag is the argument that tells Sendmail to listen on port 25 for incoming email, and this is the primary way you identify the MTA process. Sendmail maintains a queue of email that cannot be delivered immediately, and the “`-q1h`” argument tells Sendmail to start a job every hour to attempt delivery of the queued email. In fact, you can specify other time intervals if you want to process queued email faster or slower, e.g. “`-q30m`” or “`-q2h`”. I'll have some more things to say about this queue interval parameter in the *Performance Tuning* module, but in the meantime one hour is a good interval for most servers. Note that the actual location of the `sendmail` binary varies from operating system to operating system—while many Unix systems locate the binary in `/usr/sbin`, Solaris machines for example put the binary in `/usr/lib`.

The MTA process always runs as the all-powerful `root` user. This has some significant security implications, which we'll discuss in more detail in the *External Relays* module.

The primary configuration file for the MTA daemon is the `/etc/mail/sendmail.cf` file. The MTA daemon also relies on a number of other configuration files. The `/etc/mail/aliases` file (sometimes just `/etc/aliases`, depending on your preferred flavor of Unix) is where the administrator configures distribution lists, auto-responders, and other similar non-user email addresses for the system. The `/etc/mail/local-host-names` file is where the administrator keeps track of all of the different email domains and host names that this server recognizes as being “local” to this system. The `/etc/mail/relay-domains` file is used to track non-local domains and other information to properly handle email passing through this system for other destinations—we’ll come back to this file when we discuss *External Relays* later in the course.

Since I mentioned queued email earlier, I may as well tell you that the queued email being handled by the MTA process is stored in `/var/spool/mqueue`. This directory is configured to be only accessible to the `root` user, so that normal users on the system can’t read email that’s hanging around waiting to be delivered. But any administrator on the system can read the email sitting in the mail queue (again, think of email as a postcard instead of a sealed letter).



Sendmail as MSP

- Process outgoing message queue:
`/usr/sbin/sendmail -Ac -qlh`
- Runs as user `smmsp`
- Sendmail binary is SGID to `smmsp` group (put new msgs in queue)
- Config file is `/etc/mail/submit.cf`
- Queue: `/var/spool/clientmqueue`

Sendmail as MSP

There is a second `sendmail` process running on most Unix systems by default:

```
/usr/sbin/sendmail -Ac -qlh
```

The “`-Ac`” flag is what tells the `sendmail` program to behave like an MSP rather than an MTA. In other words it should read the MSP’s configuration file, `/etc/mail/submit.cf`, rather than the MTA config file (`sendmail.cf`). It should also use the MSP-specific queue directory, `/var/spool/clientmqueue`, rather than the MTA’s queue directory (`.../mqueue`). In fact, the only thing this process actually does is fire off a job every hour (“`-qlh`”) to process any undelivered email lingering in the MSP queue.

This MSP “queue runner” process has nothing to do with the normal flow of outgoing email from the system. Normally what happens is that the program that’s creating the new email message (either a user’s Unix mail client program or an automated process like the `cron` daemon) invokes the `sendmail` binary directly from the disk, and this newly executed `sendmail` process tries to immediately relay the email to whatever

MTA server the administrator has configured into the `submit.cf` file*. Only if the MTA is unreachable for some reason does the outgoing email linger in the MSP queue. Since this MTA is usually quite “close” in network geography sense to the MSP and is nearly always up and running because it’s a critical mail server, there doesn’t tend to be a lot of email hanging around in the MSP queues.

Still, you wouldn’t want normal users on the system to be able to read email that was hanging around in the MSP queue either. So the MSP queue is only readable/writable by the special “`smmmsp`” user and group that are only used by Sendmail in MSP mode. The `sendmail` binary on disk is usually set-GID to the “`smmmsp`” group so that outgoing email can be put into the queue directory as needed.

* A couple of slides ago I mentioned that the default configuration for the MSP was to submit outgoing email to the MTA process on the local machine, but that we didn’t want to do that since we wanted to shut off the MTA process on most of our Unix systems. If you’re impatient, you can actually go in and directly edit the `/etc/mail/submit.cf` file to change the default MTA, though I’ll show you a better way to handle this in just a moment. Edit the file and look for the line that reads:

```
D{MTAHost} [127.0.0.1]
```

Change the “[127.0.0.1]” to be the name of the host you want the MSP to submit outgoing email to. Save the file and you’re done!



Unix Email Lifecycle

1. Email generated in client program
2. Client program invokes `sendmail` binary directly from disk
3. Message put in `clientmqueue` while `sendmail` process contacts MTA
4. Message relayed to MTA, removed from `clientmqueue`
5. MTA stores message in `mqueue` while deciding on local vs. remote delivery:
 - Local -- aliases and user mailboxes
 - Remote -- relay or SMTP delivery

How Does This Really Work?

All of these queue directories and config files get awfully confusing and it's sometimes hard to remember the different behaviors of MSPs and MTAs. It's helpful to walk through an example of what happens to an email message from the time it gets created until the MTA injects it into the normal SMTP delivery scheme. We're going to assume for now that the email is being generated on a Unix system, because I'm more interested in describing what goes on in the Unix Sendmail MSP process than I am in what goes on inside proprietary Windows-based mail systems.

1. The email message is generated by some process on the Unix machine. It could be a user running an interactive mail client or some automated process sending email.
2. As I mentioned earlier, whatever process generated the email ends up running the `sendmail` binary directly off the local disk (I'll end up showing you the command line options for doing this in the *Sendmail Administration* module later). Because of the way it gets invoked, this `sendmail` process knows it's operating as an MSP.
3. Sendmail is very careful *never* to lose email. So the first thing this new `sendmail` process does is copy the email message into the MSP queue directory (`/var/spool/clientmqueue`). The process then attempts to contact the MTA specified in the `/etc/mail/submit.cf` file, using port 25/tcp by default.
4. If the MTA is accessible, the MSP sends the email message to the MTA using SMTP. Once the MTA indicates that it has received the entire message, the MSP closes the connection and deletes the mail message from `/var/spool/clientmqueue`.

5. Over on whatever machine the MTA is running on, however, the MTA process handling the new email message has already put the new message into `/var/spool/mqueue` directory—in fact it won't even tell the MSP that it has received the message until this happens (*very careful never* to lose email). But this MTA process is also going to try and immediately deliver the email to wherever it needs to go in order to get to its final destination. However, the MTA has to decide exactly what the “next step” is...



How Does Sendmail Know?

- Local delivery if recipient's email domain listed in `local-host-names`
 - Check `aliases` file for matching entry
 - Otherwise append to recipient's mailbox
- Otherwise follow normal outgoing email delivery rules, usually SMTP:
 - Use DNS to look up Mail Exchanger (MX) record for recipient's domain
 - Failing that, look up Address (A) record
 - Deliver email via SMTP connection to appropriate host

The MTA's Next Decision

It may seem like a very complicated decision that the MTA has to make at this point, but the reality is that the first choice is a simple: is this email local or not local? Sendmail determines this by looking at the destination address on the email—in particular the `host.domain` (or just `domain`) portion to the right of the “@” symbol in the email address.

By “local” we mean that the email is supposed to be delivered to a user on this machine. The only addresses Sendmail recognizes as “local” are the fully-qualified hostname of the machine itself (e.g., “`mailserver.foo.com`”) and the hostnames and domain names configured into the `/etc/mail/local-host-names` file by the administrator. If the righthand side of the email address matches one of these entries then the mail is “local”.

If it's a local email address then the first thing that happens is that the MTA looks up the “username” portion of the address (in other words, everything to the left of the “@” sign) in the `/etc/mail/aliases` database. If there's a match, then the MTA process does whatever the alias instructs it to do (more on aliases in the *Aliases and Mailing Lists* module at the end of the course). Otherwise, assuming the user is a valid user on the system, the mail gets delivered into that user's mailbox, which is normally called `/var/mail/<username>` (invalid user names cause an error message to be returned to the sender).

Any *other* address is by definition non-local email. Normally, the MTA process will use its normal SMTP delivery algorithm to decide where to forward this email (though I'm

going to show you lots of different ways to route email besides just relying on standard SMTP delivery).

The SMTP delivery algorithm is actually pretty simple. The MTA process does a DNS lookup using the hostname or domain name on the righthand side of the email address, and requests the special MX (“Mail eXchanger”) record for that address. The MX record is configured by the administrators that run the organization that the email is destined for and specifies which email server to use in order to get email to the given destination (the MX lookup returns both the name and the IP address of the appropriate server). For example, the email server for my own “deer-run.com” domain is a machine called “bullwinkle.deer-run.com”, so when you look up the MX record when sending email to hal@deer-run.com that’s the name you get back. The local MTA will attempt to connect to the specified host using SMTP.

If the MTA can’t find an MX record that matches the given hostname or domain name, it will try and do a DNS lookup for the IP address (“A” type) record for the given object. If it gets back an IP address, then the MTA will make an SMTP connection to the specified IP address. If the MTA can’t get either an MX record or an A record for the destination, then it kicks back an error message to the sender and throws away the email.

Assuming the local MTA is able to transmit the email to some other server as specified by the MX or A record for the destination, it then removes the message from its local queue. It’s now up to the MTA on the remote end to move the message along to its final destination. Probably the remote site has some special routing rules to get this email to where it needs to go.



What About Downed Servers?

- If mail can't be delivered immediately, it's kept in the queue
- "Queue runner" process will be started periodically to try and deliver held mail
- Queue can be "flushed" manually after a prolonged network outage
- Expect a burst of incoming email after your servers have been down

Delayed Delivery

But what if the MTA can't reach the host specified in the MX or A record? Or what if the MSP process couldn't reach its local MTA to relay the email in the first place? This is why Sendmail is so careful to queue these messages before actually attempting to forward them.

If Sendmail is unable to reach the appropriate "next hop" in the delivery chain, then the process simply exits, leaving the email in the delivery queue. After all, both the MTA process and the MSP "queue runner" process have that "-q1h" argument that tells them to attempt to deliver queued email every hour. Eventually the "next hop" server will become available and the message will be sent.

The default is for Sendmail to hold onto queued email for 5 days. While some sites may shorten this time period in order to keep their queues from getting too full, nobody in their right minds should shorten this to less than 3 days or so. After all, there could be a long weekend in the way before the administrators at a remote site come back and figure out that their mail server is down.

If you've lost Internet connectivity for an extended period and lots of email has queued up on your mail relays, you can manually start a `sendmail` process to flush the queued mail off your system (more on this in the *Sendmail Administration* module later). You should also expect a large influx of email over the first hour or two after your network connection comes back up, as all of the queued email that other sites have for your users comes flooding in. This can put a heavy load on your anti-spam and anti-virus filters, so watch out!

Note that if your Internet feed is down for so long that the mail queue directory on your external relay runs out of space, Sendmail will stop accepting new emails from your internal mail relays. The mail will not “bounce”—instead the external relay will send a “temporarily unavailable” message to the internal relays, which will in turn queue the outgoing messages until the external relay is ready to deal with them. So email backs up gracefully during an outage, but you should definitely try to avoid prolonged outages in the first place.

contract from a commercial vendor, then Sendmail Inc. (www.sendmail.com) will not only sell you the latest version of Sendmail on a nice, easy to install CD but will also allow you to give them money to provide support for it.

If you're sticking with the version you got from your vendor, one thing you will want to get from the Sendmail source distribution is the "cf" directory that contains the files you need to build Sendmail configuration files (`submit.cf` and `sendmail.cf`). Lots of Unix vendors only give you "canned" configuration files that you're supposed to hack up directly, rather than using the easier "macro style" configuration that we'll be using in this course. In this case you have to download the appropriate version of the Sendmail source and get the "cf" directory in order to recreate the examples in this course. I'll show you how to figure out what version of Sendmail you're running in the *Sendmail Administration* module a little bit later.

When downloading the Sendmail sources, it is *absolutely vital* that you check the PGP signatures on the distribution that you download. There have been cases where the bad guys have broken into the servers at `sendmail.org` and put up Trojan source distributions containing back doors that open up access on your machines. We'll cover verifying the distributions with PGP in the "hands-on" exercises at the end of this section.



Creating Sendmail Configs

- As of v8, Sendmail config files built from macro definitions
- Recommend collecting macro files into single subdir of "cf" directory

```
cd /some/path/sendmail-8.x.y/cf
mkdir myconfigs
cd myconfigs
```

- Use "m4" to create actual config files:

```
m4 myfile.mc > myfile.cf
cp myfile.cf /etc/mail/sendmail.cf
```

Creating Sendmail Configuration Files

When I first got started working with Sendmail, mail administrators wrote their `sendmail.cf` files from scratch and maintained them by hand. Here's a brief sample from the `sendmail.cf` file I use on my external mail relay for Deer Run:

```
#####
### Ruleset 0 -- Parse Address ###
#####

Sparse=0

R$*      $: $>Parse0 $1      initial parsing
R<@>    $#local $: <@>      special case error msgs
R$*      $: $>ParseLocal $1  handle local hacks
R$*      $: $>Parse1 $1      final parsing
```

The entire file is almost 2000 lines long and 60K in size. Feel like writing one of these from scratch? I didn't think so...

The good news is that all of the Sendmail v8.x series releases come with a much simpler mechanism for creating `sendmail.cf` and `submit.cf` files. Administrators can use macros to specify the configuration behaviors they want, and then run those macros through the "m4" program (a very old macro processing program for Unix) in order to produce the appropriate `.cf` file. The trick is that you need the definitions and

configuration files under that “`cf`” directory in the Sendmail source distribution as I mentioned earlier.

I find that the easiest thing to do is collect up the macro-style configuration files for all of my mail servers in a single directory. That way, when I’m trying to figure out a mail problem I can look at all of my server configs in one place. So, grab the Sendmail source distribution, verify the PGP signature, and unpack it into some appropriate directory on one of your machines (note that it need not necessarily be one of your MTAs—it could even be a laptop that you have Linux installed on). Now “`cd`” into the `cf` directory in the source distribution you unpacked. I like to make my own subdirectory for holding my configs here, but you could also keep the configs in another directory outside of the directory where you unpacked the source.

So you’ll then start creating macro configuration files using the recipes I’ll be showing you throughout the course. Normally we give these files the extension “`.mc`” to distinguish them from the “`.cf`” files you create with them, but you’re not required to do this. I usually name the file after the host that it applies to, e.g. “`external.mc`” or “`internal.mc`”, etc.

Once you’ve got a macro file all tweaked up, producing a `.cf` file is easy. To convert “`myfile.mc`” into “`myfile.cf`”, just do:

```
m4 myfile.mc > myfile.cf
```

In other words, suck the contents of `myfile.mc` into `m4` and write the output, which would normally be displayed in your terminal window, to the file `myfile.cf`. The `myfile.cf` file is then copied over to the appropriate machine (use SSH) and installed as `/etc/mail/sendmail.cf` or `/etc/mail/submit.cf` depending on whether you’re configuring the MTA or the MSP.

Exercises

1.1. Verifying and Unpacking the Sendmail Source Code

Exercise Goal – Learn how to use PGP to verify the signatures on the Open Source Sendmail distribution and how to unpack the source archive.

In your home directory you should find three files that you will need to complete this exercise:

sendmail.tar.gz – Sendmail source archive
sendmail.tar.gz.sig – "Signature file" for source archive
sendmail-key.asc – Key used to produce signature file

We will be using the GNU PGP implementation, usually called GNUPG or GPG. The gpg executable should already be installed on your system—you can verify this by running the command "gpg --help".

1.1.1. Normally you verify the signature file for a source archive by putting the signature file and the source archive together in one directory and invoking `gpg` on the signature file. What happens when you do that in this case? How would you fix the problem?

1.1.2. OK, now we need to load the key used to create the signature file. Normally, you'd use the "key ID" to find this signature at one of the "public key servers" on the Internet (like <http://pgp.mit.edu/>), but I've already grabbed the key for you. To load the key you use the command `gpg --import` on the key file I've provided. Load the key and then try and verify the signature file again. What about the warning you get from `gpg`? How would you resolve this issue?

1.1.3. Now that we've verified the software archive, we can go ahead and unpack it. See if you can figure out the correct options to the `tar` command to unpack the archive. Where is the source distribution unpacked once the `tar` command is finished?

Hint: You might want to run the command `tar --help` and look at the examples near the top of the output...

1.2. Creating a `submit.cf` File Via Sendmail Macros

Exercise Goal – Get familiar with the process for creating configuration files from Sendmail macros. Also, learn the "right" way to create a `submit.cf` file that relays outgoing email to some other mail server than the MTA running on the local machine.

To complete this exercise you will need a copy of the `submit.mc` macro file you will find in your home directory.

1.2.1. During the lecture, I suggested collecting all of your macro configuration files in a subdirectory of the "`cf`" portion of the Sendmail source distribution. Now that we've unpacked the Sendmail source distribution in your home directory, go ahead and create a directory to hold your macro files. Write down the commands you used to do this.

1.2.2. Copy "`submit.mc`" file from your home directory into your personal configuration directory and use the `m4` command to process this file and create a file called "`submit-orig.cf`". Write down the commands you use in the space below.

Hint: You can use "`~/<filename>`" to access a file in your home directory, and the character "`.`" can be used to reference the directory you're currently in.

1.2.3 In one of the footnotes in the course notes I mentioned that one way you can change where outgoing email is sent is to edit the `/etc/mail/submit.cf` file directly and change the setting of the `MTAHost` value from `"[127.0.0.1]"` to the name of the machine where you want outgoing email to end up. Looking at your copy of the `submit.mc` macro file—can you guess how to modify this file to accomplish the same goal? Modify the `submit.mc` file to direct outgoing email to a machine called `"internal.sysiphus.com"` and use `m4` to create a file called `"submit.cf"` based on your changes. Then use the `diff` command to compare the `submit.cf` and `submit-orig.cf` files—did your change to `submit.mc` do the right thing? Take notes on this process in the space below.

1.2.4. You may have also noticed in the `submit.mc` file that there is a reference to support for the DECNET networking protocol, which is frankly not used much anymore. Let's simplify our configuration a little by removing DECNET support. Rename your current `submit.cf` file to `submit-decnet.cf` and create a new `submit.cf` file without DECNET support. Run the `diff` command again to see the differences between the two files. Take notes on this process below.

Answers to Exercises

1.1. Verifying and Unpacking the Sendmail Source Code

1.1.1. Normally you verify the signature file for a source archive by putting the signature file and the source archive together in one directory and invoking `gpg` on the signature file. What happens when you do that in this case? How would you fix the problem?

Let's try running `gpg` and see what happens:

```
$ gpg sendmail.tar.gz.sig
gpg: directory `/home/.../.gnupg' created
gpg: new configuration file `/home/.../gpg.conf' created
gpg: WARNING: options in `/home/.../gpg.conf' are not
      yet active during this run
gpg: keyring `/home/.../.gnupg/secring.gpg' created
gpg: keyring `/home/.../.gnupg/pubring.gpg' created
gpg: Signature made Sun 27 Mar 2005 04:57:10 PM PST
      using RSA key ID 1EF99251
gpg: Can't check signature: public key not found
```

The first thing we see in the output is `gpg` automatically creating a bunch of files for us in the `.gnupg` directory under the home directory for our user ID. This only happens the first time you run `gpg`, when you don't already have a `.gnupg` directory.

The "interesting" stuff is happening in the last few of lines of output. `gpg` tells you when the signature file was created and the "key ID" (a unique hexadecimal string) that identifies the key used to prepare the signature file. The last line shows `gpg` complaining because it can't find the key required to verify the signature file. `gpg` can't find the key because we haven't loaded it into our "key ring" yet. We'll take a look at how to do that in the next exercise.

1.1.2. OK, now we need to load the key used to create the signature file. Normally, you'd use the "key ID" to find this signature at one of the "public key servers" on the Internet (like <http://pgp.mit.edu/>), but I've already grabbed the key for you. To load the key you use the command "gpg --import" on the key file I've provided. Load the key and then try and verify the signature file again. What about the warning you get from gpg? How would you resolve this issue?

Here's some sample output from the gpg commands you would normally run:

```
$ gpg --import sendmail-key.asc
gpg: /home/.../.gnupg/trustdb.gpg: trustdb created
gpg: key 1EF99251: public key "Sendmail Signing
      Key/2005 <sendmail@Sendmail.ORG>" imported
gpg: Total number processed: 1
gpg:             imported: 1 (RSA: 1)
gpg: no ultimately trusted keys found
$ gpg sendmail.tar.gz.sig
gpg: Signature made Sun 27 Mar 2005 04:57:10 PM PST
      using RSA key ID 1EF99251
gpg: Good signature from "Sendmail Signing Key/2005
      <sendmail@Sendmail.ORG>"
gpg: WARNING: This key is not certified with a trusted
      signature!
gpg: There is no indication that the signature belongs
      to the owner.
Primary key fingerprint: 4B 38 0E 0B 41 E8 FC 79 ...
```

The key import succeeds with no problems, and we see that when we go to verify the signature again gpg is telling us "Good signature", meaning that the signature verifies that the source code archive hasn't been tampered with. However, gpg also throws a warning at us that the key has no "trusted signature" so gpg is unable to verify "that the signature belongs to the owner".

What's going on here is that gpg is trying to tell you that perhaps you shouldn't trust this key. After all, you don't know where this key came from—I could have made it up just to spoof you into accepting an archive with a back-door Trojan. PGP uses a complicated "web of trust" to verify keys, but frankly it's "good enough" security if you get the source code and the signature file from one archive server (typically <ftp.sendmail.org>) and obtain the key file from an independent key server (like pgp.mit.edu). An attacker would have to compromise both servers in order to create and sign a malicious version of the Sendmail distribution—not impossible, but highly unlikely. Note that you can also usually find a copy of the Sendmail signing key on www.sendmail.org, and you can compare that version against the version you get from the public key server as an additional verification step if you want.

1.1.3. Now that we've verified the software archive, we can go ahead and unpack it. See if you can figure out the correct options to the `tar` command to unpack the archive. Where is the source distribution unpacked once the `tar` command is finished?

The correct command to extract tar archive files is "`tar -xf <filename>`". When the file has a `.gz` extension that means the archive is compressed ("gzip-ed"). You can specify the "z" option to automatically uncompress the archive file on the fly ("`tar -zxvf <filename>.gz`"), but the GNU `tar` command is smart enough to recognize the `.gz` extension and automatically assume this option even if you don't specify it.

Note that if you're on a system that does come with the GNU version of `tar`, your vendor's `tar` command may not be able to handle compressed archive files on its own. You will need to install the GNU `gzip` utilities, which include a program called `zcat` ("`gunzip -c`" is an equivalent command) that can be used to uncompress the archive file before feeding it to your vendor's `tar` program:

```
zcat <filename>.gz | tar -xf -
```

In the above command the output of the `zcat` command (the uncompressed archive file) is being fed in as input to the `tar` command. Note that instead of the usual file name argument to the `tar` command we're using the special "-", indicating that `tar` should read the archive file from its standard input, rather than from a file on disk.

Either way, once the `tar` archive is unpacked, you should see a new directory called "`sendmail-<vers>`", where `<vers>` is some Sendmail version string like "8.13.4". Feel free to poke around in this directory—after all, if you accidentally mess up the directory, you can always delete it and recreate it from the `tar` archive file.

1.2. Creating a `submit.cf` File Via Sendmail Macros

1.2.1. During the lecture, I suggested collecting all of your macro configuration files in a subdirectory of the "cf" portion of the Sendmail source distribution. Now that we've unpacked the Sendmail source distribution in your home directory, go ahead and create a directory to hold your macro files. Write down the commands you used to do this.

There's more than one way to do this, but here's some sample command output. Note that this example assumes you're still in the directory where you unpacked the source distribution:

```
$ cd sendmail-<vers>/cf
/home/.../sendmail-<vers>/cf
$ mkdir myconf
$ cd myconf
/home/.../sendmail-<vers>/cf/myconf
```

1.2.2. Copy "submit.mc" file from your home directory into your personal configuration directory and use the `m4` command to process this file and create a file called "submit-orig.cf". Write down the commands you use in the space below.

Again, there's more than one way to do this, but here's an example:

```
$ cp ~/submit.mc .
$ m4 submit.mc >submit-orig.cf
```

1.2.3 In one of the footnotes in the course notes I mentioned that one way you can change where outgoing email is sent is to edit the `/etc/mail/submit.cf` file directly and change the setting of the "MTAHost" value from "[127.0.0.1]" to the name of the machine where you want outgoing email to end up. Looking at your copy of the `submit.mc` macro file—can you guess how to modify this file to accomplish the same goal? Modify the `submit.mc` file to direct outgoing email to a machine called "internal.sysiphus.com" and use `m4` to create a file called "submit.cf" based on your changes. Then use the `diff` command to compare the `submit.cf` and `submit-orig.cf` files—did your change to `submit.mc` do the right thing? Take notes on this process in the space below.

Here's the contents of the original `submit.mc` file:

```
include(`../m4/cf.m4')
define(`confCF_VERSION', `Submit')
define(`__OSTYPE__', `')
define(`_USE_DECNET_SYNTAX_', `1')
define(`confTIME_ZONE', `USE_TZ')
define(`confDONT_INIT_GROUPS', `True')
FEATURE(`msp', `[127.0.0.1]')
```

As you might have guessed, the "FEATURE(`msp', ...)" line at the end of the file is where you can set the host you want outgoing email to be relayed to. Simply change the "[127.0.0.1]" to be the name of the machine where you want your outgoing email to end up:

```
include(`../m4/cf.m4')
define(`confCF_VERSION', `Submit')
define(`__OSTYPE__', `')
define(`_USE_DECNET_SYNTAX_', `1')
define(`confTIME_ZONE', `USE_TZ')
define(`confDONT_INIT_GROUPS', `True')
FEATURE(`msp', `internal.sysiphus.com')
```

Note that you don't want the square brackets around the host name—you're only supposed to have square brackets around IP addresses.

Once you've modified your `submit.mc` file, here's what your `m4` command and `diff` output might look like:

```
$ m4 submit.mc >submit.cf
```



```

$ diff submit.cf submit-orig.cf
19c19
< ##### built by ... on Tue ...
---
> ##### built by ... on Tue ...
119c119
< D{MTAHost}internal.sysiphus.com
---
> D{MTAHost}[127.0.0.1]

```

Notice that the `m4` macros that come with Sendmail create a comment in every config file showing which user ran the `m4` command and what date/time the file was created. The only other change in the file is the setting of the `MTAHost` variable—precisely what we wanted to accomplish.

1.2.4. You may have also noticed in the `submit.mc` file that there is a reference to support for the DECNET networking protocol, which is frankly not used much anymore. Let's simplify our configuration a little by removing DECNET support. Rename your current `submit.cf` file to `submit-decnet.cf` and create a new `submit.cf` file without DECNET support. Run the `diff` command again to see the differences between the two files. Take notes on this process below.

You want to delete the line from your `submit.mc` file that reads:

```
define(`_USE_DECNET_SYNTAX_', `1')
```

Once you've deleted this line, the process for renaming the current `submit.cf` file, creating a new one, and running `diff` should look something like:

```

$ mv submit.cf submit-decnet.cf
$ m4 submit.mc >submit.cf
$ diff submit.cf submit-decnet.cf
19c19
< ##### built by ... on Tue ...
---
> ##### built by ... on Tue ...
36a37
>
633a635,638
> # convert node::user addresses into a domain-based ...
[... several more lines of output not shown ...]

```

This process of making incremental changes and running `diff` afterwards to see the results is a useful technique to avoid shooting yourself in the foot too badly when making configuration changes on your mail servers.

Internal Email Relays



Internal Relays

Since the configuration of internal relay servers tends to be simpler than the configuration on your external relays, let's start with the internal servers first. After completing this module you should:

- Understand the basic syntax rules and configuration directives in Sendmail macro configuration files
- Understand how to use Sendmail's `mailertable` feature to handle special email routing rules
- Understand how to use the `makemap` program to build Unix databases for Sendmail
- Understand masquerading to hide system host names



Simple Internal Configuration

```
dn1 Configuration for internal relay server
dn1 Run this through m4 to get a .cf file
```

```
include(`../m4/cf.m4')
OSTYPE(`linux')

FEATURE(`use_cw_file')

FEATURE(`mailertable',
        `hash -o /etc/mail/mailertable')

MAILER(smtp)
```

Internal Relay Configuration With `mailertable`

Rather than pussyfoot around, let's just dive right into our first set of configuration macros. If you run these macros through `m4`, you'll end up with a configuration file that actually will function as an internal relay server for your organization. There's lots of other functionality that we'll be adding in as we go along, but since this is our first real example, I want to talk about some of the basic elements in some detail.

First let me point out some basic syntax issues. The `m4` "dn1" (Delete through New Line) directive means ignore everything after the `dn1` up through and including the newline at the end of the line. It's how you can put comments in your macro configuration files.

The other important thing to point out is that `m4` uses *balanced quotes*, which is very different from other Unix configuration files. For example in the normal Unix command shell you'd type:

```
echo 'This is quoted text'
```

Both quotes in the above expression are "right quotes", aka apostrophes. On the other hand, `m4` uses quoted expressions like:

```
OSTYPE(`linux')
```

Notice the use of the left and right quotes. In Unix-speak, the left quote is usually referred to as a "backtick".

You will also notice that it's OK to have a newline in the middle of one of our macro declarations, as you see with the two lines beginning "FEATURE(`mailetable', ..." Just be careful never to introduce a newline in the middle of a quoted string.

Now let's talk about the actual configuration macros you see here. The first line is required in all of your macro configuration files. It includes the macro definitions from the `cf.m4` file, which are what allows `m4` to expand all of the other directives in this file into usable `sendmail.cf` language. The `cf.m4` file lives in the `.../cf/m4` directory in the Sendmail source distribution. Since I told you to keep all of your macro definition files in a subdirectory of the `cf` directory, the path name `../m4/cf.m4` is the proper place to find this file, assuming you're in the directory where your macro definition files reside.

The next line defines what OS we're planning on using the resulting Sendmail configuration file on. Here we're specifying a Linux system, but obviously Sendmail also supports other operating systems like Solaris (``solaris2'`), HP-UX (``hpux10'`, ``hpux11'`, etc), and so on—Sendmail runs on a huge variety of different operating systems. The purpose of the `OSTYPE` macro is to tell Sendmail where it should look for various files and directories on a particular operating system. For example, most systems keep their aliases file in `/etc/mail/aliases`, but some older operating systems use `/etc/aliases` instead. You must specify `OSTYPE`, or `m4` will exit with an error and not produce a working configuration file.

Remember how I told you that you put any additional local host and domain names for a given machine in the `/etc/mail/local-host-names` file? This is actually not the default behavior for Sendmail—the default is to encode this information into the `sendmail.cf` file itself. But given that you tend to update this information more frequently than you change your `sendmail.cf` file, it makes sense to put your local email domains in an external file like `local-host-names`. The "use_cw_file" feature is what tells Sendmail to look for these names in the `local-host-names` file. I recommend you use this feature in all of your configuration files.

By the way, you might be wondering why the feature is called "use_cw_file" when the file you actually put things in is called `local-host-names`. The answer is that older versions of Sendmail used to call the file `/etc/mail/sendmail.cw`. But this was too easy to confuse with the `sendmail.cf` file, so the default file name changed but nobody changed the name of the feature in the macro configuration language. *sigh*

Now we come to the heart of the matter—the declarations that we will be using to control email routing on this machine. The `MAILER(smtp)` directive enables the standard SMTP delivery mechanisms. While order is normally not very important in these macro configuration files, it is critical that your `MAILER` definitions always appear at the end of the configuration file or your configuration file doesn't get properly made.

Typically, however, basic SMTP routing rules are not sufficient for internal relay servers. Most organizations end up having lots of "special case" routing needs. The `mailertable` feature is a way of creating a simple database that allows you to associate a particular email domain, like "foo.com", with a particular email server where email to "user@foo.com" should be forwarded. We will be looking at the `mailertable` database and the meaning of the somewhat obscure `FEATURE(`mailertable', ...)` declaration in much more detail on the next couple of slides.



FEATURE (`mailertable`)

- The `mailertable` feature allows email routing via a simple database:

```
sysiphus.com      smtp:exch01.sysiphus.com
eng.sysiphus.com  smtp:mail.eng.sysiphus.com
other.com         smtp:exch01.sfo.sysiphus.com
.                 smtp:external.sysiphus.com
```

- This text description must be converted into a Unix DB file after updates:
 - File you edit: `/etc/mail/mailertable`
 - File Sendmail reads: `.../mailertable.db`

The mailertable Feature

Basic Overview

You create your `mailertable` as a text file with two columns—the filename is usually `/etc/mail/mailertable`. The left column lists a domain, and the right hand column defines how and where to send email for users in that domain. Take a look at the example on the slide:

- The first line says that all mail for addresses like "user@sysiphus.com" should be sent, via SMTP, to the machine `exch01.sysiphus.com`.^{*} We'll suppose is an Exchange SMTP gateway that leads to the Microsoft Exchange servers where your users actually receive their email.
- The second line routes email for addresses of the form "user@eng.sysiphus.com" via SMTP to the host `mail.eng.sysiphus.com`. Maybe this is a Unix mail server in the Engineering department, or just their own Windows-based mail server. The point is that if you route email to multiple subdomains, you need to have an entry for each individual

^{*} `sysiphus.com` is a domain that I own just so that I can use it in teaching examples like this one—we'll be using it throughout this course. That way, if students accidentally forget to change one of the examples to use their own domain name, there's no harm done.

By the way, in Greek mythology Sysiphus was the guy who was doomed forever to push a huge boulder up a very steep hillside. Every time he got the rock to the top of the hill, it would roll off the other side and he'd have to start again. I think this is a nice metaphor for System and Network Administration.

subdomain. Of course, if you want these subdomain addresses to be something that users *outside* your company can send email to, then you're also going to have to have appropriate configuration on your external mail servers to allow this. We'll tackle this in the *External Email Relays* module coming up later.

- The third line routes email for "user@other.com" to `exch01.sfo.sysiphus.com`. Maybe the San Francisco office is another company we recently acquired, and for backwards compatibility reasons we're still routing email for their old domain name. Again, though, we're going to have to do some configuration on our external mail servers to be able to route incoming email to users in this domain from the outside world.
- The "." on the last line matches any address that doesn't match one of the other rules in the file (it turns out that order is unimportant, so you can put the "." entry anywhere in the file you want). The way this is supposed to work is that you list *all* of the email domains you use internally in the `mailertable` file, and then anything else is by definition outgoing email that needs to get forwarded up to your external mail relays.

You might wonder if we even need the `MAILER(smtp)` directive if we're using the `mailertable` to completely specify how this machine routes email—no email is actually going to be delivered using the normal SMTP routing algorithms. The `MAILER(smtp)` declaration is still required, however, since we're using SMTP as the delivery agent to all of our destinations. For modern mail servers, there will never be a case where you don't have `MAILER(smtp)` at the end of your macro definition file.

While you as the email administrator make changes and updates to the text-based `/etc/mail/mailertable` file, Sendmail wants to use a version of the file that allows for quicker lookups. So after making changes to the text file you're supposed to convert the text file into a standard Unix binary database format, usually a so-called "Berkeley DB" file. Text files are converted using a program called `makemap`, which we'll discuss in more detail on the next slide.

The conversion process creates a file called `/etc/mail/mailertable.db`, which is the file that Sendmail actually looks at. It's important to remember to rebuild the `.db` file every time you change the text file, or else your updates won't be picked up by Sendmail. Note that Sendmail re-opens the `.db` file every time it needs to handle a new email message, so there's no need to restart Sendmail after you update the `.db` file.

So now let's reconsider the `mailertable` declaration we saw on the previous slide:

```
FEATURE(`mailertable',  
        `hash -o /etc/mail/mailertable')
```

"hash" means that Sendmail should expect the `mailertable` database to be a Berkeley DB style hashed database file. The `-o` option means that this database is optional—if the `mailertable` DB file doesn't exist, Sendmail just ignores this fact and uses the standard SMTP delivery mechanisms to route email. The pathname `/etc/mail/mailertable` is the "base name" of the `mailertable` database. Since this is a "hash" style database, Sendmail expects the actual database file to be `/etc/mail/mailertable.db`.

Note that older Unix systems (like Solaris and HP-UX) will typically use a different form of database file called a DBM file. There's really very little difference from the administrator's perspective as far as using DBM files instead of Berkeley DB files. DBM style databases use *two* files—`mailertable.dir` and `mailertable.pag`—as the binary database representation of your text file. The declaration of the `mailertable` feature is also a little bit different when you're using DBM files:

```
FEATURE(`mailertable',
        `dbm -o /etc/mail/mailertable')
```

Notice the use of the keyword `"dbm"` in the above declaration, rather than `"hash"`.



Creating DB Files

- Convert text to DB file with **makemap**:

```
cd /etc/mail
makemap hash mailertable <mailertable
```

- If your OS doesn't ship with **makemap**, the source ships with Sendmail:

```
cd /path/to/sendmail-8.x.y/makemap
sh Build install
```

- Good idea to create a **Makefile** for simpler updates

Creating the `mailertable` Database

You can convert your text file into a database file using the `makemap` program. `makemap` was probably already installed when you loaded the operating system, but if your OS vendor does not provide `makemap` you can find the source code in the Sendmail source distribution. Just "cd" into the `makemap` subdirectory and run the "Build" shell script. Here the "sh Build install" runs this script via the standard Unix command shell with the "install" argument that tells the Build script to install the `makemap` binary automatically once it's been compiled. The usual installation directory is `/usr/sbin`, but that can vary from OS to OS.

Using `makemap` is straightforward: you specify the type of database to build ("hash" based on our examples, but "dbm" also works for older systems) and the "base name" of the database. `makemap` wants to get your text file as its standard input, so here we're using the "<" input redirection syntax to suck in our `mailertable` text file. When the `makemap` operation is complete, you should have a file called `makemap.db` in the current working directory.

If you don't want to have to remember the correct syntax for `makemap`, you could create a file called "Makefile" in the `/etc/mail` directory with the following lines:


```
all:: mailertable.db

mailertable.db:: mailertable
    makemap hash mailertable < mailertable
```

Note that the last line above must begin with a literal tab character and not just regular spaces.

Once this file exists, you can just `cd` into the `/etc/mail` directory type "make" to rebuild your database. The `Makefile` you created is a "recipe" that tells the `make` program how to create the `mailertable` database from the text file. The `make` program understands that it's only necessary to run the `makemap` command if the `mailertable` text file is newer than the `mailertable.db` file—that's what the second to last line of the `Makefile` describes.

Anyway, creating a `Makefile` like this is a convenient shortcut for both mail administrators and lower-level operators who may be responsible for updating the `mailertable` text file.



More Options!

```
include(`../m4/cf.m4')
OSTYPE(`linux')

define(`confMIME_FORMAT_ERRORS', `False')
define(`confMAX_HOP', `50')

FEATURE(`use_cw_file')
FEATURE(`mailertable',
        `hash -o /etc/mail/mailertable')

FEATURE(`promiscuous_relay')
FEATURE(`accept_unqualified_senders')

MAILER(smtp)
```

Other Configuration Directives

Sendmail has a reputation for being complicated to configure. So let's complicate our configuration some more!

By default, recent versions of Sendmail will format all bounce messages and other errors as MIME multipart emails. Frankly, I find plain text bounce messages a lot easier to read, so I set `MIME_FORMAT_ERRORS` to ``False'` on all my mail servers. Your mileage may vary.

Sendmail tries to stop email that gets caught in a routing loop. After the email has gone through a certain number of relays, or "hops" as Sendmail refers to them, Sendmail will drop the email and generate an error message back to the sender. The default hop limit is 18 hops, but alias expansions count as a hop, so if you have aliases that point to other aliases that point to other aliases then you really start racking up hop counts quickly. Frankly, 18 hops is usually enough, but as sites deploy more and more layers of firewalls and email relays you start to worry about the possibility of rejecting legitimate email (a so-called "false positive"). The cost of throwing an email that gets caught in a routing loop back and forth a few dozen more times is insignificant compared to the cost of a false positive, so go ahead and increase `MAX_HOP` to some larger value as we do here.

By default Sendmail has some built-in controls for restricting certain types of unwanted traffic. While these controls are *very* important on your external relay servers because of the threat of spam and other malicious behaviors, you can make life easier for yourself and other email administrators inside your company by relaxing some of these controls.

For example, `FEATURE(`promiscuous_relay')` tells Sendmail to relax and allow any host to relay email through this machine. This simplifies your configuration somewhat because you no longer have to tell Sendmail explicitly which hosts are and are not allowed to relay email through this system. However, for reasons we will discuss in the next module, you must *never* enable this feature on any mail server that is directly accessible from the outside Internet. In fact when you include this configuration directive in your macro configuration file, you see a warning every time you use the macros to build a configuration file:

```
$ m4 internal.mc >internal.cf
*** WARNING: FEATURE(`promiscuous_relay') configures
    your system as open relay.  Do NOT use it on a
    server that is connected to the Internet!
```

One trick spammers like to use is to send in email where the sender address is just an unqualified username like "eve", rather than a fully-qualified email address like "eve@sysiphus.com". These unqualified addresses can confuse users into thinking that the email came from somebody in the same organization. Normally Sendmail refuses to accept email from unqualified sender addresses, but you may discover that you have some older, broken email clients in your organization that actually emit emails with unqualified sender addresses. You can tell your internal mail relays to accept these emails by enabling the "accept_unqualified_senders" feature. Once it accepts these emails, your internal relay will automatically modify the sender address to make it fully-qualified before forwarding the email on to its final destination.

The question is what exactly will the internal relay add to make the address fully-qualified? The answer is waiting on the next slide...



Qualifying Email Addresses

- By default, Sendmail qualifies unqualified email with *host.domain*
- Usually, however, you want to "hide" the hostname portion
- This is referred to as *masquerading*:

```
MASQUERADE_AS(`sysiphus.com')  
FEATURE(`allmasquerade')  
FEATURE(`masquerade_envelope')  
FEATURE(`always_add_domain')
```

- Can put these in `submit.cf` too...

Masquerading

The default for Sendmail is to qualify any bare address with the fully-qualified hostname of the local machine. So, in our example, "eve" would normally become "eve@internal.sysiphus.com". Usually, however, you would like to hide the hostname portion and only use your local domain name.

Masquerading allows the Sendmail administrator to specify exactly what Sendmail should use when qualifying unqualified addresses. In fact, you don't have to use your local domain at all—you could use a completely foreign domain name, though I can't see a case where this would actually be useful.

In this case, we're specifying `MASQUERADE_AS(`sysiphus.com')`, so "eve" would be morphed to "eve@sysiphus.com", which is probably the correct email address for people to reply to if they want to get email to Eve. As you can see, there are also a number of other features that you also tend to turn on when you're masquerading:

- `FEATURE(`allmasquerade')` says to also apply the domain specified by `MASQUERADE_AS` to any unqualified recipient addresses for the message. For example, Eve may have used a local Unix mail client to send email to "alice" and "bob"—best if we nicely qualify everything before the mail gets any farther.
- The "masquerade_envelope" feature says to also apply the `MASQUERADE_AS` domain to addresses in the "envelope" of the message. We'll talk about envelopes in the *Sendmail Administration* section coming up later, but for now take my word for it that you want to enable this option.

- The "always_add_domain" feature means to always add the MASQUERADE_AS domain, even if users "alice" and "bob" in our previous example were local users with mailboxes on this machine.

The bottom line is you want to pick a consistent email domain name for your organization (or possibly each part of your organization if you want to use subdomain names like `eng.sysiphus.com`). Then use MASQUERADE_AS and all of its related options to enforce that domain on unqualified mail that's being generated by your users. Modern mail clients will normally emit fully-qualified email, so masquerading is becoming less critical, but it's still a good idea to configure masquerading "just in case".

So here's our "final" macro configuration file for our internal mail relays, with all of the masquerading directives included:

```
include(`../m4/cf.m4')
OSTYPE(`linux')

define(`confMIME_FORMAT_ERRORS',`False')
define(`confMAX_HOP',`50')

FEATURE(`use_cw_file')
FEATURE(`mailertable',
        `hash -o /etc/mail/mailertable')

FEATURE(`promiscuous_relay')
FEATURE(`accept_unqualified_senders')

MASQUERADE_AS(`sysiphus.com')
FEATURE(`allmasquerade')
FEATURE(`masquerade_envelope')
FEATURE(`always_add_domain')

MAILER(smtp)
```

Also, to help prevent unqualified email addresses being emitted by your Unix mail servers you should configure masquerading in the `submit.cf` files used by the MSPs on all of your Unix systems running Sendmail. Here's a simple macro configuration file for creating a `submit.cf` with masquerading enabled:

```
include(`../m4/cf.m4')
define(`confCF_VERSION', `Submit')
define(`__OSTYPE__', `')
define(`confTIME_ZONE', `USE_TZ')
define(`confDONT_INIT_GROUPS', `True')

MASQUERADE_AS(`sysiphus.com')
FEATURE(`allmasquerade')
FEATURE(`always_add_domain')
FEATURE(`masquerade_envelope')

FEATURE(`msp', `internal.sysiphus.com')
```

Exercises

2.1. Basic Configuration

Exercise Goal – To learn how to update your Sendmail configuration to set up a basic internal relay configuration, including a simple mailertable. Get some experience using the makemap command to create Unix DB files.

You should find a file in your home directory called `internal.mc`, which is a basic macro configuration file provided to save you from having to type everything in manually.

2.1.1. Copy the `internal.mc` file to the directory where you're collecting your macro configuration files. Use `m4` to create a `.cf` file and copy this file over your existing `sendmail.cf`. Write down the commands you use in the space below.

Hint: You will need to use the `/bin/su` command to become root in order to overwrite the `/etc/mail/sendmail.cf` file.

2.1.2. Now create a simple `mailertable` text file that sends all mail for users in the `sysiphus.com` domain to the machine `exchange.sysiphus.com`. All other email should be sent to the machine `external.sysiphus.com`. Use `makemap` to convert the text file into a Unix DB file. Write down the commands you use in the space below.

Hint: Remember to create this file in `/etc/mail`. You will need to be `root` to do this.

2.1.3. Modify your `submit.mc` file to add the masquerading directives we just covered. Also undo the changes we made in the last hands-on exercise so that outgoing email once again gets submitted to the "loopback" IP address (`127.0.0.1`). Create a new `submit.cf` file and overwrite the version in `/etc/mail`. Make notes on this process in the space below.

Hint: You can just cut and paste the masquerading directives from your `internal.mc` file to make things easier.

2.2. Basic Administration

Exercise Goal – Practice restarting the Sendmail processes to load your configuration updates (much more on this later in the course in the Sendmail Administration module). Also experiment with creating a Makefile to rebuild your mailertable database in a more automatic fashion.

2.2.1. Now we need to stop and start the Sendmail daemon to have it re-read our new configuration files. Use the command `"/etc/init.d/sendmail restart"` to accomplish this. Take notes on this process in the space below.

Hint: You will need to be root to perform this action.

2.2.2. Create a Makefile in the `/etc/mail` directory to make rebuilding the `mailertable` database more simple. Once you've created the Makefile, you can test it by using the command `"touch mailertable"` to update the last modified time on the `mailertable` text file so that it appears to be "newer" than the Unix DB file. Once you've done that, you should be able to run the `make` command and see the DB file being updated.

Hint: Don't forget that it's important to use tabs at certain points in your Makefile!

Answers to Exercises

2.1. Basic Configuration

2.1.1. Copy the `internal.mc` file to the directory where you're collecting your macro configuration files. Use `m4` to create a `.cf` file and copy this file over your existing `sendmail.cf`. Write down the commands you use in the space below.

Assuming you're in the directory where you're collecting your macro configuration files, the commands should look something like this:

```
$ cp ~/internal.mc .
$ m4 internal.mc > internal.cf
$ /bin/su
Password: <not echoed>
# cp internal.cf /etc/mail/sendmail.cf
```

2.1.2. Now create a simple `mailertable` text file that sends all mail for users in the `sysiphus.com` domain to the machine `exchange.sysiphus.com`. All other email should be sent to the machine `external.sysiphus.com`. Use `makemap` to convert the text file into a Unix DB file. Write down the commands you use in the space below.

The contents of your `/etc/mail/mailertable` file should look like this:

```
sysiphus.com      smtp:exchange.sysiphus.com
.                 smtp:external.sysiphus.com
```

The actual order of these two lines doesn't matter.

Assuming you're already `root` and in the `/etc/mail` directory, just run the following `makemap` command to build the database:

```
makemap hash mailertable < mailertable
```

2.1.3. Modify your `submit.mc` file to add the masquerading directives we just covered. Also undo the changes we made in the last hands-on exercise so that outgoing email once again gets submitted to the "loopback" IP address (127.0.0.1). Create a new `submit.cf` file and overwrite the version in `/etc/mail`. Make notes on this process in the space below.

When all is said and done, your `submit.mc` file should now look like:

```
include(`../m4/cf.m4')
define(`confCF_VERSION', `Submit')
define(`__OSTYPE__', `')
define(`confTIME_ZONE', `USE_TZ')
define(`confDONT_INIT_GROUPS', `True')

MASQUERADE_AS(`sysiphus.com')
FEATURE(`allmasquerade')
FEATURE(`always_add_domain')
FEATURE(`masquerade_envelope')

FEATURE(`msp', `[127.0.0.1]')
```

Don't forget the square brackets around the IP address on the last line.

Updating `submit.mc` should be straightforward:

```
$ m4 submit.mc > submit.cf
$ /bin/su
Password: <not echoed>
# cp submit.cf /etc/mail/submit.cf
```

2.2. Basic Administration

2.2.1. Now we need to stop and start the Sendmail daemon to have it re-read our new configuration files. Use the command `/etc/init.d/sendmail restart` to accomplish this. Take notes on this process in the space below.

As long as you still have `root` privileges, you should be able to:

```
# /etc/init.d/sendmail restart
Shutting down sendmail:          [ OK ]
Shutting down sm-client:         [ OK ]
Starting sendmail:               [ OK ]
Starting sm-client:              [ OK ]
```

2.2.2. Create a Makefile in the `/etc/mail` directory to make rebuilding the `mailertable` database more simple. Once you've created the Makefile, you can test it by using the command `touch mailertable` to update the last modified time on the `mailertable` text file so that it appears to be "newer" than the Unix DB file. Once you've done that, you should be able to run the `make` command and see the DB file being updated.

Per the course notes, the Makefile looks like this:

```
all:: mailertable.db

mailertable.db:: mailertable
    makemap hash mailertable < mailertable
```

Don't forget that the whitespace on the last line above must be a literal tab character, and not regular spaces.

Once you've got the Makefile in place in `/etc/mail`, database updates should look something like this:

```
# touch mailertable
# make
makemap hash mailertable < mailertable
```

Of course in normal practice you would edit the `mailertable` file and make changes. That would cause the timestamp on the file to be updated and you wouldn't normally have to use the `touch` command. We're just using `touch` here in order to force a change to the timestamp on the `mailertable` file so that we can test our Makefile.

External Email Relays



External Mail Relays

This module looks at the basic configuration you will need to create an external email relay for your organization. After completing this module you should:

- Understand the importance of OS-level security configuration for your external mail servers and have some pointers for recipes for hardening your systems
- Know what being an open relay means and know how to prevent it
- Have a basic configuration recipe for external email servers
- Know how to configure Sendmail to run as a non-root user for security
- Understand the additional configuration steps necessary beyond the basic `sendmail.cf` file

Other important issues for external mail relays, particularly spam and virus control, will be handled in later modules in the course.



System Security First!

- Your external mail relays will be targets
- Develop a "hardened" operating system configuration *before* deploying Sendmail
- Lots of free advice and tools available, including www.CISecurity.org
- Also think about using a similar config for your *internal* mail servers

Operating System Security

Your external email relay servers are some of the first machines external attackers are likely to try for. After all, the machines are directly connected to the Internet, running Sendmail (which has a well-known history of security vulnerabilities), and well advertised in your DNS database. So security on these machines is critical.

We'll talk about ways to improve the security of your Sendmail installation as we go along, but it's just as important to focus on the security of the operating system platform that you're running Sendmail on. Even if you had a completely secure Sendmail installation, attackers could still get control of the machine if you had exploitable operating system vulnerabilities (and vice-versa).

What you need to do is develop a secure or "hardened" operating system configuration for Internet-facing machines like your mail servers (and web servers, DNS servers, etc). Luckily, if your site hasn't already done this, there are lots of OS hardening "recipes" available out there on the Internet. A good place to start is the Center for Internet Security (www.CISecurity.org), which has recipes for many different Unix platforms, Windows, etc. There are also automated tools which help you tighten down the security on your Unix systems—like Bastille for Linux, HP-UX, and MacOS X (www.bastille-linux.org) or the Solaris Security Toolkit (formerly JASS) for Solaris machines (<http://www.sun.com/software/security/jass/>). Also in a Solaris vein is a hardening recipe I wrote for SANS, available from my web site: <http://www.deer-run.com/~hal/solaris-sbs/>.

While it's vital to spend lots of time up front securing your external mail relays, it's also a good idea to spend at least a little time locking down your internal mail servers as well.

For example, we have seen Sendmail exploits that are only triggered by “local delivery” (when Sendmail goes to resolve an alias or append the email to a user’s mailbox). So the exploit might pass cleanly through your external mail relays and only become a nuisance once it hits your internal mail infrastructure.



80/20 Security Checklist

- Install the smallest OS image possible
- Disable *all* unused services
- Patch regularly
- Only use SSH for admin access
- Use host- or network-based firewall
- Deploy FIA software

I teach a week long track on Unix Security for SANS, so I'm obviously not going to be able to make you experts on the subject in just a couple of slides. However, if you look at all of the different Unix security recipes available on the Internet and in books, their advice seems to boil down to the same major points:

- *Only install the minimum operating system image you need to accomplish your business mission.* For a mail relay, this is a tiny operating system image indeed. You typically don't need the Unix X Windows windowing environment because the machines tend to live in racks in locked data centers and are managed remotely. You don't need compilers and development tools because you shouldn't be coding or compiling software on these machines (compile elsewhere and copy the binaries over the network). You shouldn't use "workgroup" type tools like NFS, NIS, etc because they're rather insecure. The less stuff you load onto the machine's hard drives, the fewer potential vulnerabilities you're going to have. Also, the machine is more stable (less that can go wrong) and boots much faster (less to do at boot time). Note that you probably will want a copy of Perl on your external relays, because many of the popular Open Source anti-spam tools rely on Perl.
- *Disable all system services that are not being used.* Unix operating systems ship with a lot of different services enabled by default, even though those services may not be used by most organizations. The only thing those services are doing is acting as a potential point of entry for a determined attacker. So disable every service that you don't need—even if you can't imagine how an attacker would use that service to compromise the machine (attackers often have a better imagination than you do). If you're not sure whether you need a service or not, shut it off. If something breaks as a result of disabling a service, you can always re-enable the service later. The only

services that I run on my mail servers are Sendmail, SSH for remote access, and NTP to keep my system clock in synch with the other machines on my network (for log file timestamps) plus the usual system services like `cron`, `Syslog`, etc.

- *Keep up-to-date on vendor patches.* Security is never a static thing. New vulnerabilities are being discovered all the time and your OS vendor issues patches to address these problems. You need to come up with an automated strategy for making sure your patches are up-to-date. If you rely on administrators to do the patching by hand, then the patching won't get done. Remember that every patch you install could possibly break your system configuration, so always test your patches on a non-production machine before rolling them out more widely.
- *Use an encrypted login and file transfer mechanism (such as SSH) for administrative access.* Never use an unencrypted protocol like `telnet`, `rlogin`, or `FTP` because these protocols are susceptible to sniffing (intercepting data by reading it off the wire) and session hijacking (when a third-party breaks in and takes over your connection—usually after you've gotten `root` access on the remote machine). Don't rely on switched networks for protection: attackers have learned how to intercept and hijack sessions even on switched networks. Anyway, SSH also supports many forms of strong authentication and extra features like X Windows session tunneling, which make it much more useful than standard login/file transfer protocols.
- *Filter network traffic with a network-based and/or host-based firewall.* Your external mail relays should be protected from the Internet by a properly configured network firewall. The outside world really only needs access to port 25/tcp on your mail servers and *nothing else*. For additional security, you may even want to enable a host-based firewall (like IP Tables under Linux).
- *Use File Integrity Assessment (FIA) software and pay attention to what it tells you.* FIA software—Tripwire™, Samhain, AIDE, etc—keeps track of the current state of files in your operating system and lets you know when they've been modified. Not only does this alert you to unauthorized changes after a security incident, but it can also tell you when administrators make changes to the state of the system. In fact, the long-term benefit of FIA software is probably more as part of your change control system than as a direct security-related tool.



Now Let's Talk Sendmail

- External relays should simply pass mail between inside and outside
- Want to avoid "local delivery":
 - No user mailboxes on external relays
 - Also no alias expansions
- Also need to be careful not to become an "open relay"...

External Sendmail Posture

Just as with our operating system security principles, one of the best ways to make the Sendmail installation on our external servers more secure is to limit what we do with Sendmail on those machines. The only thing we should do with email on the external servers is relay it—either from the outside Internet to our internal mail infrastructure or from internal users out to recipients at other Internet sites.

We want to avoid the complexity of having aliases and/or user mailboxes on our external servers. These don't belong on our Internet-facing mail servers, and besides (as I mentioned earlier) there are a number of Sendmail vulnerabilities that have been triggered only when local delivery is attempted. So if we avoid local delivery, we avoid those sorts of issues.

So goal is to create external servers that are relays only. However, as I mentioned in the previous module, being too promiscuous in terms of how our external servers relay email can be an issue as well. We'll discuss this in more detail on the next slide...



No Open Relays!

- Two types of email are OK:
 - Incoming email from the Internet addressed to your email domains
 - Outbound email from your networks headed for other sites

- Accepting other types of email means you're an "open relay":
 - Spammers will use you to forward spam to other sites (consumes bandwidth)
 - You will be "blacklisted" (denial-of-service)

No Promiscuous Relays!

Your external relay servers should only allow two types of traffic:

- Incoming email, where some other site on the Internet sends you email addressed to one of your users.
- Outbound email from your user community to users at other sites

If you allow any other sort of email, then what you're doing is allowing users at other Internet sites to relay email through your servers to addresses at other organizations—this is called being a *promiscuous relay*. *Open relay* is another term that's often used to mean the same thing.

Not only is this a waste of your computing resources, but also spammers will find your mail servers and use them to relay their spam to other sites. Once that starts happening, other sites on the Internet will start to "blacklist" your mail servers and refuse to accept any email from them, making it impossible for your users to send email to many Internet sites. It is often difficult to get removed from these blacklists once you're on them.

The question is how does Sendmail correctly identify the email that it should forward as opposed to the promiscuous relay traffic?



Automatic Relay Checks

- The `local-host-names` file contains local email domains for inbound checks
- The `relay-domains` file contains:
 - IP addresses of host/networks you're willing to relay outgoing email for
 - Domains not listed in `local-host-names` that you're willing to accept email for

One way Sendmail can figure out your local email domains is by looking in the `local-host-names` file. Remember that this file contains the hostnames and domain names that Sendmail should recognize as being local to a given machine. Of course, your external mail server might also act as a relay for lots of other email domains that you wouldn't want to list in `local-host-names`—for example, if you're willing to act as a relay for email to a company that you just acquired, but which hasn't yet been integrated into your primary corporate email domain. You may list these other domains in the `relay-domains` file, and Sendmail will allow traffic to pass for these domains as well. In fact, this is the reason the `relay-domains` file exists at all—to help Sendmail decide what email is allowed to flow through this machine.

The trickier part is properly recognizing outgoing email traffic. If you did this only by the sender's domain name, you would have a big problem because spammers could simply forge their email so that it appears to come from one of your users (I'll show you how to forge email a bit later, but trust me when I tell you it's very simple). It turns out the “right” way to recognize outgoing email from your internal user community is by IP address—specifically the IP address of the remote end of the SMTP connection that's sending your external relay the email.

It turns out that the `relay-domains` file also allows you to list the IP addresses of hosts and networks you're willing to relay outgoing email for. So your `relay-domains` file might look like:

```
eng.sysiphus.com
other.com
10.1.1
10.1.2
```

This means your willing to act as a relay for email addressed to users “@eng.sysiphus.com” and “@other.com” (but not subdomains of other.com—you have to list those separately). You’re also willing to relay outgoing email as long as it comes from either of the two networks listed above.

Notice that you have to list network addresses in terms of “partial dotted quads”. In other words, you can only specify networks that are delimited on the 8-bit octet boundaries. If you’ve been granted a small IP space by your ISP, like a /28 network with only 15 usable addresses, you’ll have to list each IP address individually in `relay-domains`.

Note that we didn't have to worry about any of these issues on our internal relay server configuration because we used the "promiscuous_relay" feature to tell Sendmail to allow any host to relay email. But again, this configuration is not appropriate for machines connected to the Internet.



Simple External Relay Config

```
include(`../m4/cf.m4')
OSTYPE(`linux')

define(`confMIME_FORMAT_ERRORS', `False')
define(`confMAX_HOP', `50')

FEATURE(`use_cw_file')
FEATURE(`mailertable',
        `hash -o /etc/mail/mailertable')

define(`MAIL_HUB', `internal.sysiphus.com')

MAILER(smtp)
```

Simple External Relay Configuration

Let's take a first pass at a simple configuration for our external relay server(s). Again, we'll be adding functionality as we go along, but let's start with what we have here.

As you can see, we're using many of the same configuration directives that we used in our internal relay server configuration. You'll notice that I haven't included either the "promiscuous_relay" feature or the "accept_unqualified_senders" directive. Again, you should *never* use these features on relays that communicate with the Internet. I've also left off the masquerading directives we discussed at the end of the last module. Since the external mail relays will not be accepting email from unqualified sender addresses, masquerading is not really an issue. Outgoing email from this machine will be qualified by the masquerading directives in the `submit.cf` file. You could include the masquerading directives in the configuration you see on the slide if you wanted to—they wouldn't do any harm, though they won't do much good either.

OK, so we've got the basic stuff, but what's this `MAIL_HUB` thing? Remember how I said that we don't want any local delivery to happen on our external mail relays? Well the `MAIL_HUB` server is where this machine will send any email that would otherwise normally be delivered locally on this machine. For an email to be considered "local" in this sense it must either be addressed to an email address on this machine itself ("`user@external.sysiphus.com`", which frankly shouldn't happen) or to one of the domain names listed in the `local-host-names` file. Normally Sendmail would try and drop this email directly into the given user's mailbox on this machine, which is exactly what we don't want to happen. `MAIL_HUB` says to instead forward this email to another machine and let that machine deal with it.

It turns out that there's a kind of strange issue with this `MAIL_HUB` directive that we'll need to deal with by modifying the configuration of our *internal* mail relay. I'll come back to this problem at the end of this module, but for now let's continue discussing the configuration of our external mail relay.



What's in the `mailertable`?

- Configure all domains you want to relay to the internal mail servers:

```
sysiphus.com      smtp:internal.sysiphus.com
eng.sysiphus.com  smtp:internal.sysiphus.com
other.com         smtp:internal.sysiphus.com
```

- You must also put these domains in the `relay-domains` file
- Could have the "." directive if using an outsourced virus filtering service

The way we use the `mailertable` on the external relays is a bit different from what we saw earlier for our internal relay configuration.

Notice that the destination for all of the domains is the `internal.sysiphus.com` relay server. Your firewall policy might allow the external relay server to forward the email directly to the appropriate host for each domain, but most organizations are very restrictive about the SMTP traffic they allow through their firewalls. Also, you wouldn't want to have to update the `mailertable` on both the internal and external relays every time you changed your internal email routing rules. It's just easier to forward everything to the internal relay and let that machine figure things out. Any domain you include in your `mailertable` must also be listed in your `relay-domains` file, or Sendmail will treat any email sent to users in those domains as a promiscuous relay attempt.

You'll also notice that I haven't included a "." rule anywhere in this configuration. What happens to email that isn't addressed to one of the domains listed here? Since the behavior is unspecified by the `mailertable` database, the normal SMTP delivery mechanism takes over and non-local email gets forwarded across the Internet to the appropriate mail server. This is exactly what we want.

Some organizations are starting to use outsourced virus scanning services for incoming *and outgoing* email. In this case, we would want to force all of our non-local email through a particular set of mail servers for scanning purposes. You could add a "." rule to your `mailertable` like:

```
.      smtp:mail8.message-labs.com
```

Here we're pushing all of our outgoing email through `mail8.messagelabs.com` (MessageLabs being one of the more popular outsourcing providers these days). Obviously, you should only do this if you actually have a contract with MessageLabs and they've told you to use this mail relay.

Other Useful Additions

```
include(`../m4/cf.m4')
OSTYPE(`linux')

define(`confSMTP_LOGIN_MSG', `mailer ready')
define(`LOCAL_SHELL_PATH', `/dev/null')
define(`confPRIVACY_FLAGS',
    ``noexpn,novrfy,noverb,noetrn'')
define(`confTO_IDENT', `0s')
define(`confMIME_FORMAT_ERRORS', `False')
define(`confMAX_HOP', `50')

FEATURE(`use_cw_file')
FEATURE(`mailertable',
    `hash -o /etc/mail/mailertable')
define(`MAIL_HUB', `internal.sysiphus.com')
MAILER(smtp)
```

Additional Configuration

Now let's add some additional configuration directives that are useful on external relays. Sorry things are a bit squashed on the slide—it's getting hard to fit everything nicely into Powerpoint. Just pay attention to the highlighted configuration directives for now.

By default, when you connect to port 25 on a machine running Sendmail you see an SMTP greeting string that contains the version of Sendmail running on the remote machine:

```
220 external.sysiphus.com ESMTP Sendmail 8.13.4/8.13.4; ...
```

Now an attacker might know of a Sendmail vulnerability against a particular version of Sendmail, and if you advertise what version of Sendmail you're running you're only making the attacker's job easier. So you're better off hiding this information, at least on your external mail servers. Yes, this is "security through obscurity", but even obscurity can be a layer that makes the attacker's life more difficult. Anyway, we're changing the default greeting string with the line:

```
define(`confSMTP_LOGIN_MSG', `mailer ready')
```

The actual text of the message is unimportant as far as Sendmail and other SMTP servers are concerned, and you can make this text be anything you want. Rather than paying attention to the text, SMTP servers use the message numbers that appear at the beginning of each response from the remote machine—like the "220 ..." that appears in the sample greeting string I showed you above.

Note that you could also apply this setting on your internal mail relays. However, it's actually useful to allow your internal email administrators to connect to these machines remotely and see what version of Sendmail is running—for example, so they can figure out whether or not the machine needs to be upgraded. In the *Sendmail Administration* module I will show you another mechanism for determining what version of Sendmail is installed on a given machine if you want to hide this information on all of your mail servers.

`LOCAL_SHELL_PATH` is the path name to the Unix command shell that should be invoked when resolving an alias that executes an external program (more on this in the *Aliases and Mailing Lists* module at the end of the course). We're not supposed to have any aliases on our external servers because we're not supposed to be doing local delivery. Just to make extra special sure of that, we're setting `LOCAL_SHELL_PATH` to point to the system `/dev/null` device, which is obviously not a valid command shell. Even if we somehow did end up with an alias on this machine and we somehow did end up doing local delivery (impossible due to the `MAIL_HUB` directive), the alias would refuse to execute because the value of `LOCAL_SHELL_PATH` was invalid.

The `PRIVACY_FLAGS` option enables you to disable certain features of SMTP and Sendmail. The `noexpn`, `novrfy`, `noverb`, and `noetrn` options disable the `EXPN`, `VRFY`, `VERB`, and `ETRN` SMTP commands respectively:

- `EXPN` is an SMTP command that expands an alias and gives back a list of the email addresses that are members of that alias. Spammers can use this to get a list of valid email addresses by connecting to port 25 on a mail server and issuing repeated `EXPN` commands (although email administrators can also use it to debug problems with aliases). We shouldn't have any aliases on our external servers anyway, but we may as well disable this feature.
- `VRFY` is similar to `EXPN` except that it allows the remote user to verify a single email address as being valid on the remote machine. This can be used by attackers to find valid user accounts on the system as part of a break-in attempt.
- `VERB` is an SMTP command that puts the SMTP session into "verbose" (debugging) mode. This can give away extra information about your mail server and the machines it routes email to in your environment, which again could be useful to an outside attacker. Best to turn this off as well.
- `ETRN` is an extended SMTP (ESMTP) command that allows a remote site to request that you send them any queued email you might have for them. In the ISP world, `ETRN` is often used by dialup users to tell the ISP's mail server that their connection is up and they're ready to receive email. On other mail servers, however, it's a potential denial-of-service attack because malicious users can barrage you with bogus `ETRN` requests.

By the way, notice that when the argument to an `m4` macro is a comma-separated list as we have here, you are supposed to use doubled single quotes around this argument. And yes, you have to use balanced quotes here as you see in the example.

The Ident protocol (RFC 1413) was designed to allow a server like Sendmail to get information about the remote machine that's making a connection. The problem is that on the modern Internet everybody blocks Ident at their firewalls, and many sites drop the Ident packets silently, meaning Sendmail must wait (5 seconds by default) before it actually accepts the connection. Since Ident is useless on the modern Internet, we tell Sendmail not to bother doing Ident queries by setting `TO_IDENT` to zero seconds. You may also want to add this setting to the configuration file for your internal relay server, just so you don't have to bother with the Ident queries there either.



Do I Have to Run as `root`?

- Why the Sendmail MTA runs as `root`:
 - Bind to port 25/tcp
 - Perform "local delivery" for different users
 - Read and write config files and queues
- But...
 - Can give up privs after binding to 25/tcp
 - No local delivery on external mail server
 - Can change permissions on files and queue

*So at least on our external machines,
maybe we don't have to run as `root`...*

Limiting Sendmail Privileges

Why Does Sendmail Run as `root`?

One of the knocks on Sendmail from a security perspective is that the MTA daemon that's listening on port 25 is running with `root` privileges. The problem there is that if there's a remotely exploitable vulnerability in Sendmail, then the attacker instantly gets `root` privileges, which means that they can do anything at all on the system. So why does Sendmail run as `root`, or perhaps more importantly is there a way we can run Sendmail as some other unprivileged user?

Sendmail runs as `root` for several reasons:

- Only processes running as `root` are allowed to bind to network ports below 1024, at least on Unix machines (on Windows systems, for example, any user can bind to any port). Since the MTA daemon needs to bind to port 25/tcp, it needs to be `root`.
- When Sendmail is performing local delivery, it needs to be `root` so that it can append email to different user's mailboxes.
- Various Sendmail configuration files and directories, like the aliases database and the mail queue directory, are only accessible to the `root` user. Sendmail must run as `root` to access this information.

However, there are work-arounds for all of the above issues—at least on our external relay server:

- If you look at other servers on Unix that need to bind to low port numbers—for example, the BIND DNS server—the daemon will run as `root` only for as long as it needs to bind to the low port number. Thereafter it will give up `root` privileges and run as an unprivileged user.
- We are not doing any kind of local delivery on our external mail relay, so appending mail to user mailboxes is not an issue for us.
- While you don't want many of Sendmail's critical files and directories to be readable by normal users on the system, that doesn't mean those files and directories have to be owned by the `root` user. We can create a special user just for Sendmail, and make everything owned by that user instead.

It's actually pretty straightforward to get all this set up and working on your external relay servers—you could even use this configuration on your internal relay servers as long as you're not doing local delivery there. There's just a little bit of system reconfiguration and then a new configuration directive in your Sendmail macro definition file. Just follow along with me on the next couple of slides.



System Configuration

- Create new `sendmail` user/group
- Shut down Sendmail MTA daemon
- Reset permissions on critical directories:

```
chown -R sendmail:sendmail /var/spool/mqueue
chgrp -R sendmail /etc/mail
chmod -R g+r /etc/mail
chmod g+s /etc/mail
```

- Restart Sendmail MTA daemon with new config, setting `RUN_AS_USER`

System Configuration

The first step is to create a new user ID and group ID that your Sendmail MTA daemon will run as. *Do not* use the special "smmsp" user and group that the MSP uses—create a new user and group ID. I will often create a user and group called "sendmail" with UID and GID 24 (smmsp is usually UID and GID 25). Nobody will ever actually log in as this user, so you can lock the password entry and use an invalid shell and home directory. Here are some sample commands you can use for setting up this user and group, assuming your system supports the traditional `useradd` and `groupadd` commands:

```
groupadd -g 24 sendmail
useradd -u 24 -g 24 -M -d /var/spool/mqueue \
-s /dev/null sendmail
```

Note that you need to create the new group first, because we're going to use that group as one of the arguments to the `useradd` command. As you can probably guess, the "-u" and "-g" options are used to specify the user ID and group ID for the new user and group. The "-d" option specifies the home directory for the new user and the "-M" option tells the `useradd` command not to create this directory (since it already exists). The "-s" option specifies the default command shell for the user—here `/dev/null` is an invalid shell to help prevent user logins via this account.

Now we're going to start messing around with ownerships and permissions of various files and directories. Before you do that, however, it's a good idea to shut down the running Sendmail daemon so that it doesn't get confused while you're in the middle of

making your changes. We'll talk in more detail about stopping and starting Sendmail in the *Sendmail Administration* module, but on most systems the commands `"/etc/init.d/sendmail stop"` or `"pkill sendmail"` will work.

There are really two critical directories for the MTA process: the queue directory and the `/etc/mail` configuration area. The queue directory should be owned by whatever user and group you created to run the MTA daemon as (`"sendmail"` and `"sendmail"` in our example). The permissions on this directory don't need to be changed—it's already mode 700 so only `root` can access the files in there, we're just changing which user is allowed to peek.

You still want the `/etc/mail` directory and the files in it to be owned by `root`, because you only want legitimate system administrators to be messing around with these files. But we need to make sure that the files in this directory are at least readable by the MTA daemon when it's running unprivileged. So what we're going to do is change the group ownership on the directory and its contents to our `"sendmail"` group, and then use `"chmod -R g+r /etc/mail"` to give group read permissions to everything in the directory. Actually, the files in there are probably already group readable but it never hurts to be sure.

The last `"chmod g+s /etc/mail"` command adds the so-called "set-GID" bit onto the `/etc/mail` directory. On Unix systems, if set-GID is set on a directory, then any new file created in that directory will automatically inherit the group ownership of the directory (as opposed to the group membership of the user that creates the file, which would be the default). In this way we can make sure that any newly created files end up with the proper ownerships—like when you're rebuilding the `mailertable` database with the `makemap` program.

Note that if you're on a platform that keeps the aliases database in the `/etc` directory instead of `/etc/mail`, then you'll want to run the command `"chgrp sendmail /etc/aliases*"` in addition to the commands you see on this slide. You'd need to issue the same command every time you rebuilt the aliases database, but since we don't expect to have aliases on our external relay, this shouldn't be an issue.

Now we're ready to tweak our Sendmail configuration and fire up Sendmail again...

RUN_AS_USER Option

```
include(`../m4/cf.m4')
OSTYPE(`linux')
define(`confRUN_AS_USER', `sendmail:sendmail')
define(`confSMTP_LOGIN_MSG', `mailer ready')
define(`LOCAL_SHELL_PATH', `/dev/null')
define(`confPRIVACY_FLAGS',
    ``noexpn,novrfy,noverb,noetrn,noreceipts'')
define(`confMIME_FORMAT_ERRORS', `False')
define(`confMAX_HOP', `50')
define(`confTO_IDENT', `0s')
FEATURE(`use_cw_file')
FEATURE(`mailertable',
    `hash -o /etc/mail/mailertable')
define(`MAIL_HUB', `internal.sysiphus.com')
MAILER(smtp)
```

The RUN_AS_USER Option

Sendmail allows you to specify an alternate user and group to run as with the `RUN_AS_USER` configuration option. As you can see, we're specifying the "sendmail" user and group we created per the instructions on the previous slide.

If you look at the running MTA process after you set this option, you may be surprised to see that it's still running as `root`—this is expected behavior. The master MTA process always runs as `root`, but when a new SMTP connection comes in the master MTA process must create a copy of itself (*forking* in the Unix parlance) to handle the incoming connection. The "child" process will run as the unprivileged user and group you specify with the `RUN_AS_USER` option. So while the master MTA process itself runs as `root`, outsiders will only ever be able to communicate with unprivileged child processes.

Anyway, once you've run your new macro definitions through `m4`, copy the resulting config file out to your external server(s) and install it as `/etc/mail/sendmail.cf`. Now restart Sendmail—"`/etc/init.d/sendmail start`" should work on most Unix systems, or see the instructions in the *Sendmail Administration* module. You can test the configuration by using `telnet` to connect to port 25 ("`telnet localhost 25`"). If you get a process listing in another window, you should see a `sendmail` process running as the `sendmail` user. Close the `telnet` session to make this process go away.



OK, But What *Else...*?

- Populate `relay-domains` file
- Do some testing!
- Is your internal mail relay working?
- Set up MXes for you domain(s) when you're ready to get Internet email:

```
sysiphus.com. IN MX 10 external.sysiphus.com.
```

Other Considerations

Don't forget to add any domains listed in your `mailertable` file to `/etc/mail/relay-domains`. You actually shouldn't need to put anything into the `local-host-names` file—we don't want to recognize any additional domains as being "local" to this machine.

Your `relay-domains` file should also contain the IP addresses of the internal networks that will be sending outgoing email through your external relays. It might be that you've set things up so that only your internal mail relays are allowed to send email out through your external servers. In this case, the `relay-domains` file only has to list the IP addresses of your internal relay server(s). However, most sites find that it's more flexible to list all of their internal IP address space in the `relay-domains` file, just in case they decide to change their internal email routing architecture. It's your call.

The next thing you should do is test your configuration. We'll talk about testing and debugging in some detail in the *Sendmail Administration* module, but for now you can use one of the Unix mail client programs (like the "mail" command) on your external relay to try sending email to various internal and external destinations. As I alluded to earlier, the `MAIL_HUB` directive requires that we make a slight tweak to our internal mail relay configuration (more on that on the next slide), but you should at least be able to send email to the internal domains listed in the `mailertable` file.

Anyway, once you're satisfied that your external mail relay is ready to rock and roll, you can tell the world about it by changing the MX records for the domains you want to get email for. The slide shows a simple MX record for the domain "sysiphus.com". This

record says to send all email of the form "user@sysiphus.com" to the machine external.sysiphus.com (you will also need to publish an A record that gives the IP address for this machine). If you also want to get email for subdomains like "user@eng.sysiphus.com" or for completely foreign domain names like other.com, you must also publish MX records separately for those domains and subdomains. You are allowed to route email for some other domain like other.com to your external.sysiphus.com relay—the target of the MX record does not have to be in the same domain.

The number next to the MX record is a priority value. The lower the priority, the more preferred the mail server is. Sometimes organizations will have a "primary" external mail server that would normally get all of their email (listed at priority 10, say) and a "secondary" email server (listed at priority 100 maybe) in case of failure on the primary. I've seen this happen in companies that have two different divisions with separate email domains and Internet connections—each division lists itself as the primary MX for its own domain, but is willing to accept email for the other division and route it over the company's internal WAN.

The tricky thing about these backup MX servers is that they might get email even when your primary server is up and connected to the Internet. Internet routing problems, problems at other organizations, etc may make it appear to some other company like your primary mail server is down, causing them to route email to your secondary mail server. Spammers also like to try and route email through your backup MX servers because sometimes sites aren't as careful with the spam filtering on their secondary MX hosts. So your secondary MX server needs to be able to accept and process email for your domains at any time.

Notice the trailing "." at the end of "sysiphus.com." and "external.sysiphus.com."? This is a very important piece of syntax in your DNS configuration files. If you don't have these trailing dots, the standard Unix BIND name server automatically appends your domain name onto the end of all un-dotted names. This, of course, would yield nonsensical results, so don't forget the trailing dots!



That MAIL_HUB Issue

- **MAIL_HUB** will emit recipient addrs as `user@internal.sysiphus.com`

- Add **domaintable** feature to our existing internal relay config:

```
FEATURE(`domaintable',  
        `hash -o /etc/mail/domaintable')
```

- Use **domaintable** to map addresses to our canonical domain:

```
internal.sysiphus.com    sysiphus.com
```

- We'll also be using **domaintable** later for virtual email domains...

MAIL_HUB Oddity

The problematic thing about the `MAIL_HUB` directive we're using is that when it forwards the "local" mail to the `internal.sysiphus.com` mail relay it also transforms the recipient address to the form "`user@internal.sysiphus.com`". In other words, `MAIL_HUB` appends the name of the relay server to the bare username to create the final recipient address.

It turns out that this causes problems for us because the internal mail relay is not currently configured to handle email to these kinds of user addresses. In fact, what's going to happen is that the internal mail relay is going to think the email address is "local"—it is, after all, addressed to the fully-qualified hostname of our internal relay machine—and try and deliver it into the user's mailbox. This is definitely not what we want. It seems like what we'd like to do is intercept this email before local delivery and transform the "`@internal.sysiphus.com`" part into something sensible—like our canonical `sysiphus.com` domain for example.

Sendmail does include a feature for doing precisely this, called the `domaintable`. As you can see, the declaration for the `domaintable` feature is nearly identical to the `mailertable` declaration we saw earlier. And generally, you administer the `domaintable` just like your `mailertable`: edit a text file called `/etc/mail/domaintable` and then use `makemap` to transform this text file into a Unix DB file for Sendmail to read.

The `domaintable` database itself is much simpler than the `mailertable`, however. The lefthand column of the `domaintable` lists the domain you want to transform, and

the righthand column lists the domain you want to change it to. All we need on our internal relay server at the moment is the one-line `domaintable` you see on the slide that maps `internal.sysiphus.com` to `sysiphus.com`.

While the `domaintable` handles the problem we're currently experiencing, this is not actually what this feature is intended for. I'll show you more about the normal use of the `domaintable` feature in the module on *Virtual Domains* later in the course.

Exercises

3.1. Basic Configuration File Updates

Exercise Goal – Just a basic introduction to the process of generating and installing a new `sendmail.cf` file, plus the rudiments of stopping and starting the `sendmail` process to pick up your configuration changes.

You should find a file in your home directory called `external.mc`, which is a basic macro configuration file provided to save you from having to type everything in manually.

3.1.1. Suppose the local email domain is `sysiphus.com` and the company uses the network `192.168.2.0/24` internally. Create an appropriate `mailertable` that routes incoming email for the local email domain to `internal.sysiphus.com`, and also create a `relay-domains` file for this configuration. Write down the contents of each file in the space below.

3.1.2. Now copy the `external.mc` file into the directory where you're collecting your macro definition files. Use the `m4` command to generate a file called `external.cf` from this macro file. Then copy the `external.cf` file so that it replaces the `/etc/mail/sendmail.cf` file and restart the Sendmail daemon. Write down the commands you used in the space below.

3.1.3. Check to see if the Sendmail daemon is using your new configuration file by connecting to port 25 on the local machine and looking at the default greeting string. Compare this with the setting of `SMTP_LOGIN_MSG` in the `external.mc` file—do you see anything odd? Can you explain this discrepancy?

Hint: Try "telnet localhost 25" to connect to port 25 on the local machine.

3.2. Configuring Relay With RUN_AS_USER

Exercise Goal – To allow you to practice setting up Sendmail to run in unprivileged mode on relay servers, and to give you some experience verifying that your configuration is working.

3.2.1. Shut down the running Sendmail daemon and then follow the instructions in the course book to configure the file and directory permissions appropriately for our RUN_AS_USER configuration. Write down the commands you use in the space below.

[Exercises continue on next page...]

3.2.2. Now modify the `external.mc` file and add the `RUN_AS_USER` option. Recreate the `external.cf` file and copy the new file over your existing `/etc/mail/sendmail.cf` file. Restart the Sendmail process when you're done. Use the space below to make notes on the commands you use.

3.2.3. Verify that the `RUN_AS_USER` setting is behaving as you expect. The easiest way to do this is to connect to port 25 on the local machine and leave that connection open. Then, in another window on the same machine, run the command `"ps -ef | grep send"` in order to see all the Sendmail-related processes. If you see a process that's running as the `sendmail` user, then the `RUN_AS_USER` setting is working. Make notes on the processes you see in the space below.

3.3. Getting Ready to Route Some Email

Exercise Goal – Update our `submit.mc` configuration for the external relay, and add the `domaintable` feature on our internal relay so that we have a completely functional mail environment.

In the last exercise below you'll be using the standard Unix command-line mail client to send some test emails. Until we get to the Sendmail Administration module, however, you probably won't be able to tell whether things are working or not. Don't despair—the Sendmail Administration module is coming up next.

3.3.1. If you haven't done so already, update the `submit.cf` file on this external relay machine with the `submit.cf` file we created in the last hands-on exercise. The new `submit.cf` file should include our standard masquerading directives and be configured to send outgoing email to the MTA running on the local machine via the loopback address (127.0.0.1).

3.3.2. Modify your `internal.mc` configuration to include the `domaintable` declaration we covered at the end of this section, then update the configuration of your internal mail relay with the new configuration file. Create a `domaintable` that maps `internal.sysiphus.com` email to `sysiphus.com` email. Take notes on this process in the space below.

3.3.3. On a machine that is configured as an external relay, run the command `"mail mary@sysiphus.com"`. Use `"Test from <machinename>"` as the `"Subject:"` line and the body of the message. Enter a `."` on a line by itself to terminate the message and just hit `<return>` when prompted for the `"Cc:"` line—meaning we don't want to `"Cc:"` anybody on the message. Now follow a similar process on a machine that's configured as an internal relay, except this time send a piece of outgoing email to the address `hal@deer-run.com`. Take notes in the space below.

Answers to Exercises

3.1. Basic External Configuration

3.1.1. Suppose the local email domain is `sysiphus.com` and the company uses the network `192.168.2.0/24` internally. Create an appropriate `mailertable` that routes incoming email for the local email domain to `internal.sysiphus.com`, and also create a `relay-domains` file for this configuration. Write down the contents of each file in the space below.

Your `mailertable` file is only a single line:

```
sysiphus.com      smtp:internal.sysiphus.com
```

Once you've created the `mailertable` text file, don't forget to run the `makemap` command to create the `mailertable.db` file:

```
makemap hash mailertable < mailertable
```

The `relay-domains` file should list the `sysiphus.com` domain and the internal network number for the site:

```
sysiphus.com  
192.168.2
```

3.1.2. Now copy the `external.mc` file into the directory where you're collecting your macro definition files. Use the `m4` command to generate a file called `external.cf` from this macro file. Then copy the `external.cf` file so that it replaces the `/etc/mail/sendmail.cf` file and restart the Sendmail daemon. Write down the commands you used in the space below.

Assuming you're in the directory where you're collecting your macro configuration files, the commands should look something like:

```
$ cp ~/external.mc .  
$ m4 external.mc > external.cf  
$ /bin/su  
Password: <not echoed>  
# cp external.cf /etc/mail/sendmail.cf  
# /etc/init.d/sendmail restart  
Shutting down sendmail:           [ OK ]  
Shutting down sm-client:         [ OK ]  
Starting sendmail:               [ OK ]  
Starting sm-client:              [ OK ]
```

3.1.3. Check to see if the Sendmail daemon is using your new configuration file by connecting to port 25 on the local machine and looking at the default greeting string. Compare this with the setting of SMTP_LOGIN_MSG in the external.mc file—do you see anything odd? Can you explain this discrepancy?

Here's some sample output from a telnet session:

```
$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
220 mailer ESMTP ready
quit
221 2.0.0 localhost.localdomain closing connection
Connection closed by foreign host.
```

The external.mc file sets SMTP_LOGIN_MSG to be "mailer ready". So why does the greeting string appear as "mailer ESMTP ready"? ESMTP stands for Extended SMTP and is a newer version of the SMTP protocol that supports lots of additional features over standard SMTP. MTAs like Sendmail that can handle the newer ESMTP protocol are supposed to identify themselves by putting the word ESMTP in the initial greeting message. If you do not put "ESMTP" in your SMTP_LOGIN_MSG, Sendmail will automatically insert the word into this message by itself because it wants to make sure that all of the remote servers know that it's capable of handling ESMTP.

3.2. Configuring Relay With RUN_AS_USER

3.2.1. Shut down the running Sendmail daemon and then follow the instructions in the course book to configure the file and directory permissions appropriately for our RUN_AS_USER configuration. Write down the commands you use in the space below.

Here's a sample of what your session will look like:

```
# /etc/init.d/sendmail stop
Shutting down sendmail:           [ OK ]
Shutting down sm-client:         [ OK ]
# groupadd -g 24 sendmail
# useradd -u 24 -g 24 -M -d /var/spool/mqueue \
    -s /dev/null sendmail
# chown -R sendmail:sendmail /var/spool/mqueue
# chgrp -R sendmail /etc/mail
# chmod -R g+r /etc/mail
# chmod g+s /etc/mail
```

There isn't too much output to give you confirmation of what you're doing. Commands like "grep sendmail /etc/passwd /etc/shadow" and "ls -l /etc/mail" can help confirm that things are set the way you expect them to be.

3.2.2. Now modify the external.mc file and add the RUN_AS_USER option. Recreate the external.cf file and copy the new file over your existing /etc/mail/sendmail.cf file. Restart the Sendmail process when you're done. Use the space below to make notes on the commands you use.

You want to add a line somewhere in the middle of your external.mc file that reads:

```
define(`confRUN_AS_USER', `sendmail:sendmail')
```

Recreating the external.cf and replacing the sendmail.cf file works just like the previous exercise:

```
$ m4 external.mc > external.cf
$ /bin/su
Password: <not echoed>
# cp external.cf /etc/mail/sendmail.cf
# /etc/init.d/sendmail start
Starting sendmail:           [ OK ]
Starting sm-client:         [ OK ]
```

3.2.3. Verify that the `RUN_AS_USER` setting is behaving as you expect. The easiest way to do this is to connect to port 25 on the local machine and leave that connection open. Then, in another window on the same machine, run the command `"ps -ef | grep send"` in order to see all the Sendmail-related processes. If you see a process that's running as the `sendmail` user, then the `RUN_AS_USER` setting is working. Make notes on the processes you see in the space below.

In one window on the machine you're configuring, just do the `"telnet localhost 25"` command we've used previously. Leave this session running.

In another window on the same machine, you should see `ps` output that looks something like this:

```
# ps -ef | grep send
root      3324      1 0 13:40 ?        00:00:00 sendmail:...
smmsp     3330      1 0 13:40 ?        00:00:00 sendmail:...
sendmail  3348    3324 0 13:40 ?        00:00:00 sendmail:...
root      3362    3349 0 13:40 pts/4   00:00:00 grep send
```

We'll talk about what all of these processes are in the *Sendmail Administration* module coming up later. For now, though, notice the third line of `ps` output with the word "sendmail" in the first column? The first column indicates what user the process is running as, so it would appear that our `RUN_AS_USER` setting is functioning properly.

You can close your `telnet` session in the other window once you've verified that things are working.

3.3. Getting Ready to Route Some Email

3.3.1. If you haven't done so already, update the `submit.cf` file on this external relay machine with the `submit.cf` file we created in the last hands-on exercise. The new `submit.cf` file should include our standard masquerading directives and be configured to send outgoing email to the MTA running on the local machine via the loopback address (127.0.0.1).

This process should be becoming pretty automatic for you at this point, so I won't go into a lot of detail here:

```
$ m4 submit.mc > submit.cf
$ /bin/su
Password: <not echoed>
# cp submit.cf /etc/mail/submit.cf
# /etc/init.d/sendmail restart
Shutting down sendmail:           [ OK ]
Shutting down sm-client:         [ OK ]
Starting sendmail:                [ OK ]
Starting sm-client:               [ OK ]
```

[Answers to Exercises continue on next page...]

3.3.2. Modify your `internal.mc` configuration to include the `domaintable` declaration we covered at the end of this section, then update the configuration of your internal mail relay with the new configuration file. Create a `domaintable` that maps `internal.sysiphus.com` email to `sysiphus.com` email. Take notes on this process in the space below.

Your new `internal.mc` file should look something like this:

```
include(`../m4/cf.m4')
OSTYPE(`linux')

define(`confMIME_FORMAT_ERRORS',`False')
define(`confMAX_HOP',`50')

FEATURE(`use_cw_file')
FEATURE(`mailertable',
        `hash -o /etc/mail/mailertable')
FEATURE(`domaintable',
        `hash -o /etc/mail/domaintable')

FEATURE(`promiscuous_relay')
FEATURE(`accept_unqualified_senders')

MASQUERADE_AS(`sysiphus.com')
FEATURE(`allmasquerade')
FEATURE(`masquerade_envelope')
FEATURE(`always_add_domain')

MAILER(smtp)
```

The process for updating your configuration should be old hat by now:

```
$ m4 internal.mc > internal.cf
$ /bin/su
Password: <not echoed>
# cp internal.cf /etc/mail/sendmail.cf
# /etc/init.d/sendmail restart
Shutting down sendmail:           [ OK ]
Shutting down sm-client:         [ OK ]
Starting sendmail:                [ OK ]
Starting sm-client:               [ OK ]
```

[answer continues on next page...]

The `/etc/mail/domaintable` file is a simple one-line file:

```
internal.sysiphus.com      sysiphus.com
```

Don't forget to use `makemap` to build the DB file:

```
makemap hash domaintable < domaintable
```

3.3.3. On a machine that is configured as an external relay, run the command `"mail mary@sysiphus.com"`. Use `"Test from <machinename>"` as the `"Subject:"` line and the body of the message. Enter a `"."` on a line by itself to terminate the message and just hit `<return>` when prompted for the `"Cc:"` line—meaning we don't want to `"Cc:"` anybody on the message. Now follow a similar process on a machine that's configured as an internal relay, except this time send a piece of outgoing email to the address `hal@deer-run.com`. Take notes in the space below.

Here's some sample output showing how things might look during the session on the external relay:

```
$ mail mary@sysiphus.com
Subject: test from <machinename>

test from <machinename>
.
Cc: <return>
```

Things should look almost identical when you do the test from your internal relay, so I won't bother repeating myself.

As I mentioned in the *Exercise Goals*, it's not going to be immediately obvious to you if things are working properly or not. Once everybody has finished this exercise, we'll take a look at some of the log output to see what's happening as a result of these test emails before we get into further detail in the *Sendmail Administration* module.

Sendmail Administration



Sendmail Administration

Clearly knowing how to properly configure Sendmail is important. But just as important is knowing how to manage Sendmail once you've got it configured and operating. After completing this module you should:

- Know how to figure out what version of Sendmail is installed on a machine
- Understand Sendmail's mail queue: how to see what's in it, and how to manually flush messages out of the queue
- Be able to locate the running Sendmail processes on a Unix machine and stop and start Sendmail from the command line
- Understand the information in mail headers and the Sendmail logs
- Have tools for testing both your running Sendmail configuration, as well as new configuration files you create—*before* you put those configurations into production

As a side effect of learning how to test Sendmail, I'm also going to show you how to forge email. But you must promise to use this knowledge only for good, and not for evil!

What Version of Sendmail?

```
$ /usr/sbin/sendmail -d0.1 </dev/null
Version 8.13.4
  Compiled with: DNSMAP HESIOD HES_GETMAILHOST LDAPMAP
                LOG MAP_REGEX MATCHGECOS MILTER
                MIME7TO8 MIME8TO7 NAMED_BIND NETINET
                NETINET6 NETUNIX NEWDB NIS PIPELINING
                SASLv2 SCANF SOCKETMAP STARTTLS
                TCPWRAPPERS USERDB USE_LDAP_INIT

===== SYSTEM IDENTITY (after readcf) =====
  (short domain name) $w = internal
  (canonical domain name) $j = internal.sysiphus.com
  (subdomain name) $m = sysiphus.com
  (node name) $k = internal.sysiphus.com
=====

Recipient names must be specified
```

What Version of Sendmail is Installed?

There are a couple of different ways to figure out what version of Sendmail is installed and running on a particular machine. As I showed you earlier, if the MTA daemon is actually listening on port 25, the default Sendmail greeting string shows the Sendmail version number:

```
$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
Escape character is '^]'.
220 internal.sysiphus.com ESMTP Sendmail 8.13.4/8.13.4; ...
quit
221 2.0.0 internal.sysiphus.com closing connection
Connection closed by foreign host.
```

Why is the version number reported as "8.13.4/8.13.4"? The first set of numbers is the version number of the Sendmail binary. The second set of numbers is the version number of the Sendmail distribution that was used to create the `sendmail.cf` file that the MTA daemon is running from. Sendmail tends to be very backwards compatible, and sites are often reluctant to modify their configuration once they've got a working version. So sometimes you can see some very old configuration file version numbers, even though the version of the binary is up-to-date.

But what if the MTA daemon is not running on the machine, or what if the site has used the `SMTP_LOGIN_MSG` parameter to change the default greeting to hide the Sendmail version number? If you run Sendmail in debugging mode with the `-d` flag per the example on the slide, you will see the Sendmail version number.

The "-d0.1" option turns on Sendmail's least verbose debugging level.* Normally, Sendmail would expect to have at least one recipient email address specified on the command line (hence the "Recipient names must be specified" error at the end of the output) and to get the text of the email message on its standard input (which is why we're doing the "< /dev/null" at the end of the command line on the slide). But here, however, we're just interested in seeing the Sendmail version number on the first line of output.

Of course, you also get some other interesting information. You can see all of the different compile-time options that were built into this Sendmail binary (though we won't go into detail about what any of these mean in this course). You can also see what Sendmail thinks is the canonical hostname and domain name of this machine, which can be useful when you're trying to debug Sendmail problems.

* Though we don't have time to cover them here, Sendmail supports an incredible number of different debugging levels. In the O'Reilly *Sendmail* book, the complete descriptions of all of the debugging levels for Sendmail cover almost 40 pages!

What's in the Queue?

```
# /usr/sbin/sendmail -bp
                        /var/spool/mqueue (37 requests)
-----Q-ID----- -Size- --Q-Time--- --Sender/Recipient--
jBLHvTar021527      262 Wed ... 09:57 <bounce@lamers.org>
                        (host map: lookup (lamers.org): deferred)
                        <mary@sysiphus.com>
jBHKqB8k002934      202 Sat ... 12:52 <bob@sysiphus.com>
                        (Deferred: Connection timed out with spam.com.)
                        <msgbot@spam.com>

[... 35 other entries not shown ...]
                        Total requests: 37
```

■ Add "-Ac" flag to look at MSP queue

The Sendmail Queue

What's in the Queue?

One of the ways in which Sendmail problems make themselves known is when email starts backing up in your mail queues. You can get information about the messages in the MTA's mail queue (`/var/spool/mqueue`) with `sendmail -bp`. And again the `-Ac` flag is what tells Sendmail to go into MSP mode, so `sendmail -bp -Ac` shows you what's in the MSP queue directory (`/var/spool/clientmqueue`). On most Unix systems, the `mailq` command is equivalent to `sendmail -bp`. You must run these commands as `root` to be able to get information about the mail queues.

Each message currently sitting in the mail queue generates three lines of output. On the first line you see the unique queue ID for the message (which is just a random string generated to keep queue entries from colliding with each other), the size of the message in bytes, the date and time the message was added to the queue, and the address of the sender of the message. The second line gives you information about why the message is stuck in the queue—what you're seeing is the error message from Sendmail for the last delivery attempt of the message. The third line displays the intended recipient(s) of the message. There is also information at the beginning and end of the output showing the total number of items in the queue.

So in the example on the slide, the first message in the queue is coming from the email address `bounce@lamers.org` and is trying to get to `mary@sysiphus.com`. However, we're getting this weird `Host map: lookup` error message—what's up with that? We'll diagnose this problem a bit more on the next slide. The second queue

entry in our example is a message from "bob@sysiphus.com" to "msgbot@spam.com", where we're getting a "connection timed out" error message. You tend to see this a lot when users like Bob set up an "out of the office" auto-responder on their mailbox and then get spam email from some domain. Generally, the spammers don't run mail servers that accept the return email,* so the messages get stuck in your mail queues until your Sendmail process reaches its queue timeout (5 days by default). The "connection timed out" message means that your local Sendmail daemon is trying to reach the remote mail server but getting no response.

As you can see, there are a total of 37 messages in this mail queue. Is that a lot? Should we be worried about it? Well, if 35 of the messages are from Bob's auto-responder to various spam sites, then it's probably not a big deal (although you'll probably be wanting to find a better spam filtering mechanism for your company). The point is that different servers will tend to have different amounts of email collected in their mail queues and have vastly different reasons for why that email is there. You have to learn what's "normal" for your mail servers by keeping an eye on your mail queues from day to day. You might even set up an automated job that tracks the number of messages in your mail queue and sends you an alert if the number of queued messages suddenly increases—say by more than 10% in an hour.

Of course the error message in the output of "sendmail -bp" may not be enough information for you to figure out why exactly the email can't be delivered. Often there is more information in Sendmail's log files, as we'll see on the next slide.

* Spammers who accept return email usually find themselves victimized by denial-of-service attacks, where other sites on the Internet email them large numbers of huge messages in order to take down the spammers' mail servers.



Check That Log Entry

■ Looking for a particular queue ID:

```
# grep jBLHvTar021527 /var/log/maillog
Dec 21 10:26:05 external sm-mta[32633]:
jBLHvTar021527: to=<mary@sysiphus.com>,
delay=00:28:35, xdelay=00:00:01, mailer=relay,
pri=176149, relay=internal.sysiphus.com.
[10.1.1.10], dsn=4.0.0, stat=Deferred: 451 4.1.8
Domain of sender address bounces@lamers.org does
not resolve
```

■ At this point you have to figure out if it's your problem or theirs...

Queue IDs and the Sendmail Log

The unique queue ID number displayed by "sendmail -bp" is useful for getting further information about the message from Sendmail's log file. Each entry in the log file for a particular message is tagged with the queue ID number, so you can use the `grep` command to easily pull out all of the relevant information about a particular message. Note that if Sendmail has tried repeatedly to deliver an email message, you may get quite a bit of output from the `grep` command because each attempt will generate at least one line of log output.

The next trick is actually finding the Sendmail log on your particular flavor of Unix. On many Unix systems the file is `/var/log/maillog`, but Solaris machines for example use `/var/log/syslog` instead. The easiest way to figure out where your mail logs are going is to look in the `/etc/syslog.conf` file and find the entries that deal with logs to the "mail" facility. For example:

```
mail.debug          /var/log/syslog
```

Note also that while the current log file may be `/var/log/syslog`, you will also typically find files like `/var/log/syslog.0`, `/var/log/syslog.1`, etc that contain older log entries.

OK, so now let's see if the Sendmail logs can tell us any more about that queued message that was getting the "Host map: lookup" error message. First we get the queue ID of the message from the output of "sendmail -bp", and then we `grep` for that queue ID from the Sendmail logs. There's quite a bit of information in the matching log entry,

which we'll talk about in more detail later in this section. For now, though, we're interested in the more detailed error message at the end of the log entry. As you can see, Sendmail is complaining that the domain name of the sender's address "does not resolve". In other words, Sendmail is unable to look up this domain in DNS for some reason. Sendmail will not deliver email unless it can convince itself that the sender's domain name is valid by doing a DNS lookup—this is to help stop email from spammers who just make up domain names when sending out their spam.

So why is Sendmail unable to look up this domain via DNS? It could be that you've lost Internet connectivity or your local DNS servers are hosed in some way that's preventing normal DNS query behavior. However, if you can resolve other ".org" domain names (slashdot.org, fsf.org, eff.org, etc) then the problem is probably not at your site, and so it's therefore the remote domain that's having problems. If it's the other site's fault, there's not much you can do except wait and hope they get things figured out (if their DNS is hosed up, there's probably no way for you to send them an email to tell them their DNS is hosed).



Flushing the Queue

- Flush the whole darn thing:

```
/usr/sbin/sendmail -v -q
```

- Flush selectively by recipient:

```
/usr/sbin/sendmail -v -qR@sysiphus.com
```

- Or avoid a particular recipient:

```
/usr/sbin/sendmail -v -qR\!@lamers.org
```

- Can also use "-qS" to select messages by sender or "-qI" to use queue ID

Flushing the Queue Manually

If you've had a prolonged network outage, there may be quite a bit of email backed up in your mail queues. Once connectivity is restored, you'd like to push that email along as fast as possible. The "-q" option tells Sendmail to process its queue, and as you might guess "sendmail -q -Ac" tells Sendmail to flush the MSP queue. I like to add the "-v" option so I can see what Sendmail is doing:

```
# /usr/sbin/sendmail -q -v
Running /var/spool/mqueue/jBLHvTar021527 (sequence 1 of 37)
lamers.org: Name server timeout
<bounce@lamers.org>... Transient parse error -- message queued
for future delivery
lamers.org: Name server timeout
lamers.org: Name server timeout
<mary@sysiphus.com>... Connecting to internal.sysiphus.com...
220 inernal.sysiphus.com ESMTP mailer ready at Wed, ...
>>> EHLO external.sysihpus.com
250-internal.sysiphus.com Hello external.sysiphus.com...
250-ENHANCEDSTATUSCODES
250-EXPN
250-VERB
250-8BITMIME
250-SIZE
250-ONEX
250-ETRN
250-XUSR
250 HELP
>>> MAIL From:<bounce@lamers.org> SIZE=2507
451 4.1.8 Domain of sender address bounce@lamers.org does not
resolve
```

```
<mary@sysiphus.com>... Deferred: 451 4.1.8 Domain of sender
address bounce@lamers.org does not resolve
```

```
Running /var/spool/mqueue/jBLI9wRW000389 (sequence 2 of 37)
[...and so on ...]
```

What you're seeing here is the actual SMTP communication between `external.sysiphus.com` and `internal.sysiphus.com` as `external.sysiphus.com` tries to forward the email inwards to its final destination. Toward the end of the sample output you can see the actual "Domain of sender address...does not resolve" error message that ends up in the Sendmail logs (per the previous slide). We'll have some more to say about internal SMTP communications a bit later in this module when I show you how to forge email.

One problem you run into when you've got a lot of email stacked up in the queue is getting hung up by timeouts to unreachable sites. For example, there are a bunch of bounce messages from Bob's "out of the office" auto-responder to various spam sites but you don't want to have to wait for each one of those connections to time out while you're trying to push other email through. The `sendmail -q` command allows you to be selective about which emails to process.

Suppose the internal mail relays went down and there's a bunch of email queued up on the external servers waiting to be delivered to users in the `sysiphus.com` domain. The command `sendmail -qR@sysiphus.com` would tell Sendmail just to try and deliver the emails to "`user@sysiphus.com`" addresses, which should now flow through very quickly. The "R" after the "-q" stands for "recipient".

Similarly, suppose we know that `lamers.org` is hosed up but we still want to process all of the other queued mail—"sendmail -q\!R@lamers.org" means to skip all of the email to users in the `@lamers.org` domain, but process everything else. The "!R" means "not this recipient", but since "!" is a special meta-character in the Unix command shell you have to put a backslash in front of it for the command to be parsed properly.

Of course in our examples so far the "`bounce@lamers.org`" address was the *sender* of the email, not the recipient. The `sendmail -q` command also let's you filter by sender (use "S" instead of "R") and even by individual queue IDs ("I") if you just want to manually flush a single message. So to skip all of that email in our queue from the senders in "`@lamers.org`", we would run the command `/usr/sbin/sendmail -q\!S@lamers.org -v`.

The other problem with manually mucking around in the queue is that you run the risk of removing a file while Sendmail is in the middle of attempting to deliver that message. This can cause all sorts of locking problems and really mess up your mail queues. So my first piece of advice is to avoid manual shenanigans in your mail queues, but if you just have to fiddle around with things in there make sure you shut down Sendmail first so you don't end up causing problems for yourself.

So I guess that means I better show you how to stop and start Sendmail, huh?

Sendmail Processes

```
# ps -ef | grep send
root      1628      1 0  11:43 ?        00:47:22 sendmail:
           accepting connections
smmsp    1634      1 0  11:43 ?        00:12:11 sendmail:
           Queue runner@01:00:00 for /var/spool/clientmqueue
sendmail 31500  1628 0  14:22 ?        00:00:10 sendmail:
           ./jBHKqB8k002934 spam.com.: user open (sendmail)
sendmail 31878  1628 0  14:23 ?        00:00:01 sendmail:
           server mail.example.net [145.66....] cmd read
root      31898  31883 0  14:23 pts/2  00:00:00 grep send
```

- MTA process is `root`-owned process with PPID 1 (look at third column)
- Stop this process to prevent new email from getting in from outside

Stopping and Starting Sendmail

Identifying Running Sendmail Processes

The first trick to starting and stopping Sendmail is knowing whether or not Sendmail is actually running on your machine and also what it's doing (is an MTA daemon running or just the MSP queue runner? etc). The usual command for listing processes under Unix is "ps -ef" ("ps auxww" if you're on a BSD system), and we filter the output through `grep` so we see just the Sendmail-related processes.

The first thing you might notice about this output is that the process names are things like "sendmail: accepting connections" when we actually started the Sendmail processes with command lines like `/usr/sbin/sendmail -bd -q1h`. On many Unix systems, a process is allowed to change its own process name listing in the output of `ps` (though there are plenty of main-stream operating systems like Solaris where this is not allowed). Sendmail takes advantage of this in order to provide information to the administrator about what each process is doing.

The first line of output where we see the process listed as "sendmail: accepting connections" represents the MTA process. On a Solaris machine this process would show up as `/usr/sbin/sendmail -bd -q1h`. You can also tell that this is the MTA process because it's running as the `root` user—MSP processes run as user `smmsp`, and after our `RUN_AS_USER` hack, the "child" MTA processes will run as user `sendmail`. The other clue that this is the master MTA daemon is that the "parent

process ID" (PPID) listed in the third column of the output is "1".* "Master" daemon processes will always have a PPID of 1, whether the daemon is Sendmail, Apache, or any other Unix daemon. Anyway, if your email configuration is broken or if you want to stop the MTA process so you can manually fiddle around in the `mqueue` directory then this is the process you need to shut down.

The next process tells us that it's the `"Queue runner@01:00:00 for /var/spool/clientmqueue"`. The `"clientmqueue"` portion tells us that this is an MSP-related process (`"-Ac"`), as does the fact that the process is running as the `smmsp` user. And we can also see that it's processing the queue every hour (`"-q1h"`). So this must be the MSP queue runner that we normally start with `"/usr/sbin/sendmail -Ac -q1h"` (and again, this is exactly what you'd see in a process listing on a Solaris machine).

The next two processes are interesting. Notice that both are running as the `sendmail` user, meaning that they are "child" MTA processes that are affected by our `RUN_AS_USER` setting. Now on a Solaris machine you wouldn't have much extra information to go on—the process names would be duplicates of the master MTA process' command line—but here Sendmail is telling us exactly what each process is doing. The first of these two processes is currently running the queue. You can see that it's in the middle of processing a specific queue ID, and even see the recipient address for the message. The second process is dealing with a piece of incoming email—notice that Sendmail shows you the hostname and IP address of where the connection is originating. Of course, you have to get lucky and happen to capture the process output at just the right moment to see these kinds of processes, but just in case you ever do see them you'll know what they are.

The last line of output is not actually a Sendmail process at all. It's the `grep` command that we piped the output of `ps` into in order to filter the output. You can ignore this line.

* Actually the BSD version of the `ps` command doesn't display the PPID by default, though you can force the command to show you this information with either the `"j"` or `"l"` options. But these options force the output of the `"ps"` command to display two lines of output per process, and when you pipe the output into `grep` you don't get the results you're looking for because the PPID is displayed on a different line from the process name that's being matched with `grep`.



Stopping Sendmail

- Many different options:

```
/etc/init.d/sendmail stop  
pkill sendmail  
kill 1628 1634
```

- You can still flush the queue manually even when daemons are shut down
- Use "`kill -HUP <pid>`" if you just want MTA to re-read config files

Stopping Sendmail

On many Unix machines, the command `/etc/init.d/sendmail stop` is the correct way to gracefully shut down the Sendmail processes (both the MTA and the MSP queue runner). BSD systems do not provide scripts in `/etc/init.d` for stopping and starting services, so you're going to have to do this manually instead (and don't even get me started about how Solaris 10 handles this).

Most Unix systems provide a `pkill` command that lets you manually stop processes by process name. So `pkill sendmail` is another way of stopping the Sendmail processes on the machine. Of course you might accidentally interrupt a Sendmail process that is in the middle of receiving email from a remote site, but the remote site will simply queue the email and deliver it later, so no harm done.

If your Unix operating system doesn't provide either an `/etc/init.d` script or the `pkill` command, then you have to do things "the old-fashioned way". Run the `ps` command I showed you on the previous slide to list the Sendmail processes. The process ID (PID) of each process appears in the second column. Now use those PIDs as arguments to the `kill` command to manually stop the MTA and MSP queue runner processes.

Remember that the MTA process is really only responsible for handling new incoming email messages. If your mail queue is in danger of filling up, you can shut off the MTA process and still use `sendmail -q` to manually flush messages out of the queue, even when the MTA process is not operating. You can also use this technique when you're decommissioning an old mail server—shut off the MTA process to stop getting

new emails, but use `sendmail -q` to manually flush out any queued email before shutting the machine down forever.

Most of the time you don't particularly want to shut down the Sendmail processes, you're just trying to update the MTA with new configuration information. A more graceful way to do this is to use the `ps` command to find out the process ID of the Sendmail MTA process and then do `kill -HUP <pid>` (for example, `kill -HUP 1628` if we're going by the example on the previous slide). The `kill -HUP` here means send the special "hang-up" signal to the process. Most Unix processes take this as a signal to re-read their configuration files. Sendmail actually goes a bit farther than this—the original MTA process goes away and is replaced by a new one which re-reads its configuration files as it's coming up. After the "HUP" you should see that the MTA process has a different PID.



Starting Sendmail

- Gracefully:

```
/etc/init.d/sendmail start
```

- Manually:

```
/usr/sbin/sendmail -L sm-mta -bd -q1h
```

```
/usr/sbin/sendmail -L sm-msp-queue -Ac -q1h
```

- Use of "-L" option and time interval for queue runner varies widely

Starting Sendmail

The scripts in `/etc/init.d` are capable of both stopping and starting various processes. So on most Unix systems you can easily start Sendmail simply by running `/etc/init.d/sendmail start`. Actually on Linux systems you can even stop and start the Sendmail daemons with a single `/etc/init.d/sendmail restart` command, which is useful when you're installing a new `sendmail.cf` file for example. On other Unix systems you're going to have to issue both the "stop" and "start" commands separately, however.

If you're on a BSD system or other Unix flavor that doesn't have `/etc/init.d` scripts, you can always start the daemons from the command line. The normal invocation for the MTA daemon is:

```
/usr/sbin/sendmail -bd -q1h
```

The normal invocation for the MSP queue runner is:

```
/usr/sbin/sendmail -Ac -q1h
```

Note that the location of the `sendmail` binary can vary—`/usr/sbin/sendmail` is common, but (as I mentioned earlier) Solaris uses `/usr/lib/sendmail`. Also, different sites use different values for the "-q" option, though "-q1h" is fairly common.

Notice that the examples on the slide add the "-L" option. "-L" allows the administrator to specify a process name that will be used instead of "sendmail" when Sendmail is

writing information to its logs (it has no effect on the process name that's displayed in the output of `ps`). This allows you to more easily distinguish between messages from the MTA daemon and the MSP processes in your logs. Using `-L` is not required, but it is a common option.



Helpful Testing Tools

- Sometimes when you're testing, it helps to have the "outsider" perspective
 - *Sign up for a free email account so you can send yourself test emails*
- Having "outside" Internet feed that you can easily test from is hugely useful
 - *An external machine with Unix shell access via SSH is usually enough*
- Testing labs help before big rollouts
 - *VMWare, UML, Xen, etc can help out here*

Testing Sendmail

Some Tips Before We Begin

If you're administering Sendmail, you spend an awful lot of time testing and debugging various strange issues. The weirdest issues almost always involve your communications with external sites, and when you're installing new configurations you always want to verify that inbound and outbound mail to and from other sites on the Internet is functioning. So it's really nothing short of a requirement that the people who are administering Sendmail for an organization have email accounts in one or more external email domains, just so that they can perform basic validation. Also, when you're having email troubles at your primary site and you need to communicate with outside entities (your ISP, the InterNIC, etc), you're going to need an external email address that you can send and receive email from.

One of my favorite diagnostic techniques involves using `telnet` to connect to port 25 on a running mail server and manually impersonate another SMTP server. And sometimes you want to do this from an IP address that's not listed in your external mail relay's `relay-domains` file. I like having an outside machine that I can SSH into from wherever I am on the Internet so that I can act like an "outsider" when I'm debugging mail problems. Of course, I have the machines on Deer Run's networks for when I'm working at customer sites, but I also have another account (an old University account that the head of the CS Department lets me keep in return for free consulting from time to time) that I use when troubleshooting my own mail servers.

When you're doing a major upgrade, you'd really like to be able to test your new architecture before changing the configuration on your production servers. Having a

testing lab where you can "mock up" the new architecture is a big win. If you can't afford lots of hardware to build a fully-outfitted testing lab, remember that there are "virtual machine" technologies available—VMWare, User-Mode Linux (UML), Xen, etc—that allow you to create even an entire virtual network of systems on a single hardware platform. This is great for testing new architecture ideas.

Forging Testing Email

```
# telnet external.sysiphus.com 25
[...]
220 mailer ESMTP ready
HELO spoof.example.com
250 external.sysiphus.com Hello real.dom.com [130.55....]
mail from: Im@lying.com
250 Im@lying.com... Sender ok
rcpt to: john@sysiphus.com
250 john@sysiphus.com... Recipient ok
data
354 Enter mail, end with "." on a line by itself
From: Santa@Northpole.com
To: naughty@somewhere.com
Subject: Ho Ho Ho

You've been a naughty boy this year!
.
250 KAA06575 Message accepted for delivery
```

Forging vs. Testing Email

It turns out that the common techniques for testing your email configuration also allow malicious users to easily forge email. For example, it's often useful for an email administrator to create a fake email that appears to come from somebody at another domain (so you can test how your mail architecture handles external email), but spammers and other naughty folks can use the same techniques to try and cover their tracks. So what I want to show you on the next several slides is not only techniques that you can use for testing your own mail systems, but also the kinds of nasty spoofed traffic that malicious users can create and how to detect different forms of bad behavior.

One of the most flexible testing techniques in your arsenal is the old "telnet to port 25" trick. Earlier I showed you how to use this technique to get the Sendmail version number from a machine that's running an MTA daemon (and hasn't used SMTP_LOGIN_MSG to hide this information). However, because SMTP is a simple text-based protocol you can actually pretend to be a remote SMTP server and manually inject entire email messages using this technique. This gives you complete control over the sender and recipient addresses and the entire message content, so you can easily test lots of different scenarios (internal email, external email, promiscuous relay attempts, etc). Of course, the bad guys can also use this technique to pretend to be sending their spam from your domain, which can end up causing you all kinds of problems. And the bad news is that there really isn't that much you can do at the moment to prevent this.

Take a look at the example on the slide (the stuff that you're supposed to type is in bold face, while the responses from Sendmail and other programs are in regular type face):

- First we telnet to `external.sysiphus.com` on port 25. You'll see a bit of chatter from the `telnet` client (which I don't show here) and then the greeting message from the remote Sendmail daemon.
- The first thing that happens in an SMTP communication is that the host that's making the connection (that's you) should introduce itself to the remote MTA. You do this with the "HELO" SMTP command (the basic SMTP commands are all four letters long, which is why it's HELO and not "HELLO")—you say "HELO *<hostname>*", where *<hostname>* is the fully-qualified hostname of your machine.* Alternatively you can make up a hostname, though you'll notice in the response from the remote MTA it knows your IP address and does a DNS lookup to get the canonical hostname associated with that IP. As we'll see in a moment, remote admin can see both the fake name that you HELO-ed with as well as your IP address and the hostname associated with that IP in DNS.
- Now that you're done saying HELO, you can start composing an email message. The first thing you have to do is specify the sender of the message with the "mail from:" command.† As you can see, I'm completely making up the sender address, although Sendmail will reject sender addresses that don't come from a valid DNS domain.
- Next you specify the recipient(s) of the email message with the "rcpt to:" command. If you want the email to go to multiple recipients, you enter multiple "rcpt to:" commands each with a single recipient address—so 40 recipients means 40 "rcpt to:" commands. Sendmail validates each recipient address separately, so if you make a typo that causes Sendmail to reject one of your recipients you can keep going and the email will still get delivered to the other valid recipients.
- When you're done entering recipients, it's time to send along the actual email message itself. Begin by entering the "data" command—as the remote MTA says in its response message, you then enter the text of the email message and finish it off with a "." on a line by itself. Notice that you have to include both the headers and the body of the email message after the "data" command. There must be a blank line between the headers and the body of the email—this is how all SMTP-compliant email software distinguishes the headers from the body. As you can also see in our example, the addresses in the "From:" and "To:" headers in the email message need bear no relation to the addresses you specified earlier with the "mail from:" and "rcpt to:" SMTP commands. More on this in just a second.

* Actually, it's not strictly necessary to HELO, but all proper MTAs will do so. Because spammers often use bulk-mailing software that doesn't obey normal standards and therefore doesn't HELO, there are many MTAs that are configured to reject email from remote machines that don't HELO at the beginning of the SMTP session. So it's always a good idea to HELO.

† By the way, SMTP is case-insensitive so capitalization doesn't matter.

- Once you've finished manually entering the email message, just enter a "." on a line by itself and the remote MTA accepts the message and forwards it on to be delivered. At this point you can either type "quit" to terminate the SMTP session, or you can start in with another round of "mail from:", "rcpt to:", and "data" commands to fire in more email messages.

"Envelope" vs. "Message"

The addresses you enter with the SMTP "mail from:" and "rcpt to:" commands are referred to as the *envelope* of the message. These are the addresses that Sendmail (and other MTAs) use when deciding how to route and deliver the email message. As far as Sendmail is concerned, the email message itself (the stuff you enter after the "data" command) is just a big blob of data that doesn't need to be parsed at all. Think of the real world analogy: what matters is the delivery address on the outside of the envelope. The Post Office never sees (and doesn't care about) the address that appears on the letter inside the envelope, and in fact you can accidentally stuff a letter for one person into an envelope addressed to another person.

That's why it doesn't matter what the "From:" and "To:" headers in the message say—Sendmail totally ignores this information when it comes to routing and delivering the message. This is also why your users can get spam messages delivered to their mailboxes even when their email address doesn't appear in the "To:" header. This is also how the "Bcc:" feature in most email systems works.

When we were talking about masquerading earlier (hiding your hostnames and only using the local domain name to qualify email), I mentioned that you wanted to enable the "masquerade_envelope" feature. Now you can probably guess what this feature does. In addition to masquerading the address on the "From:" line in the email message itself, "masquerade_envelope" says to also masquerade the so-called "envelope from" address (the address that gets specified with the SMTP "mail from:" command). And since we also enabled the "allmasquerade" feature, we'll also be masquerading the recipient addresses—both "To:" and the envelope address that appears in the "rcpt to:" line(s).

"Received:" headers you should believe are ones that were created by hosts under your control—anything prior to that is hearsay at best.

Take a look at the lower of the two "Received:" headers in our example—this is the one that was added by the `external.sysiphus.com` server when we were forging our email (note the "Received: ... by external.sysiphus.com..."):

- Notice that the header shows not only the bogus hostname we HELO-ed with, but also follows that up with the IP address that you connected from plus the hostname that Sendmail resolved from DNS using that IP address. So you can lie all you want, but somebody who knows how to read mail headers can figure out what you were doing.
- You also see the "john@sysiphus.com" email address that was specified in the "rcpt to:" command (and of course later on you see the bogus "To:" address that was entered after the "data" command).
- Another useful item in the "Received:" header is the queue ID that this message used on `external.sysiphus.com` ("...with ID jBM2Nc9l020")—you can use this ID to pull further information out of the Sendmail logs on `external.sysiphus.com`.

Similarly, the other "Received:" header shows you the queue ID that was used on the `internal.sysiphus.com` relay. This is useful for checking the logs on that machine.

There's one subtle bit of information in the "Message-Id:" header that might escape your notice if you didn't know to look for it. The "Message-Id:" header is normally stuck onto the message by the mail client that creates the initial message. However, if Sendmail ever receives an email message without a "Message-Id:" header, the MTA process will add one. The "Message-Id:" headers created by Sendmail always include the hostname of the machine where the "Message-Id:" was created, and here we see that the "Message-Id:" was tacked on by the machine `external.sysiphus.com`. This implies that the message was not generated by a proper mail client, and thus is probably either a forgery or spam. Of course the bad guys could fake this header like anything else, but why would they want to fake a "Message-Id:" header that in this case so obviously marks the mail as being invalid?

Checking the Logs

```
# grep jBM2Nc91020 /var/log/maillog
Dec 21 18:24:31 external sm-mta[27230]: jBM2Nc91020:
  from=Im@lying.com, size=108, class=0, nrcpts=1,
  msgid=<2005122202.jBM2Nc91020@external.sysiphus.com>,
  proto=SMTP, daemon=Daemon0, relay=real.dom.com
  [130.55...]
Dec 21 18:24:40 external sm-mta[27230]: jBM2Nc91020:
  to=john@sysiphus.com, delay=00:00:20,
  xdelay=00:00:09, mailer=relay, pri=30108,
  relay=internal.sysiphus.com. [10.1.1.10], dsn=2.0.0,
  stat=Sent (jBM2OVn23830 Message accepted for
  delivery)
```

- Don't see what they HELO-ed with, but does show total number of recipients...

Reading Sendmail Logs

Much of the same information that you see in the message headers is also available in the Sendmail logs. As we discussed earlier, you can track down the exact location of the Sendmail logs on your system by looking at your `/etc/syslog.conf` file, but you can often find these logs in `/var/log/maillog`.

Once you find the log files you then need to be able to find the log entries that match the particular message that you're investigating. If you have the "Received:" headers from the message then it's easy to track things down using the queue ID number as we're doing on the slide. Failing that, however, you're probably going to have to `grep` through the logs using either the sender or recipient of the email message and correlating that information with the time the message was received—a rather annoying process. You might also be able to use the message ID in the message to track down the log entries.

The other important fact here is Sendmail makes two log entries for each message that passes through the system: one that describes where the message came from and one that describes where the message is going to. If you `grep` for the sender address or the recipient address you're only going to get one of these entries—make sure you pull the queue ID off of that line and use the queue ID to grab the other matching entry out of the log file (if the mail server is processing a lot of messages simultaneously, the two log messages may not appear back-to-back).

The first log entry for a message gives information about where Sendmail got the message from. In this log entry you can see the envelope from address, the message ID, and the IP address and DNS name of the server that made the SMTP connection. You

don't see the hostname that this server HELO-ed with unfortunately, though you do see the total number of recipients the message was sent to. The second log entry gives information about where the email message was sent. Here we can see the address(es) specified with the "rcpt to:" commands, along with the hostname and IP address of the next hop server, and even the queue ID of the message on that machine if the remote server is running Sendmail (not all MTAs will divulge this information).

Cmd Line Version (plus "-v")

```
# /usr/sbin/sendmail -v -Am -f Im@lying.com john@sysiphus.com
From: Santa@Northpole.com
To: naughty@somewhere.com
Subject: Ho Ho Ho

You've been a naughty boy this year!
^D
john@sysiphus.com... Connecting to [10.1.1.10] via relay...
220 mailer ready
>>> EHLO external.sysiphus.com
250-internal.sysiphus.com Hello external.sysiphus.com...
>>> MAIL From:<Im@lying.com> SIZE=108
250 2.1.0 <Im@lying.com>... Sender ok
>>> RCPT To:<john@sysiphus.com>
250 2.1.5 <john@sysiphus.com>... Recipient ok
>>> DATA
354 Enter mail, end with "." on a line by itself
>>> .
250 2.0.0 jBM2gAn23892 Message accepted for delivery
john@sysiphus.com... Sent (jBM2gAn23892 Message accepted...)
Closing connection to [10.1.1.10]
>>> QUIT
221 2.0.0 internal.sysiphus.com closing connection
```

Testing from the Command Line

The "telnet to port 25" trick is not the only way of generating email for testing (and forgery) purposes. You can accomplish almost exactly the same thing using Sendmail from the command line. In the example on the slide we use the "-f" option to specify the envelope from address and then specify as many recipient addresses at the end of the command line as we want. The "-Am" flag says to act as an MTA (use the configuration in /etc/mail/sendmail.cf) rather than as an MSP (submit.cf).^{*} The "-v" option is extremely useful because it causes Sendmail to display the SMTP session used to transmit the message once you've finished entering the message headers and body (i.e., the un-bolded output in the bottom half of the slide). This gives you a lot of insight into how your Sendmail configuration thinks messages to the specified recipient(s) should be delivered.

Once you've entered the your sendmail command line, Sendmail waits for you to enter your message on the standard input. Type carefully, because you can't go back and edit the previous line once you hit return. As in our "telnet to port 25" example you enter the headers of the message, then a blank line, then the message body. You can terminate the message by entering a "." on a line by itself, or you can just hit <ctrl>-D as I'm doing here.

Now the glory of "-v" takes over. If you look carefully you can see the same SMTP commands happening as we manually entered via the telnet session in the previous

^{*} Actually, you may recall that "-Ac" tells Sendmail to operate in MSP mode (be a "client"). The "-A" argument followed by *any other letter* tells Sendmail to operate as an MTA. It's just that "-Am" is traditional.

example. Actually, if you look really closely you'll see that Sendmail is issuing an EHLO command instead of HELO. This is not a typo. EHLO is the ESMTP (Extended SMTP) version of the HELO command and is used to indicate that the client is capable of speaking ESMTP and not just the old standard SMTP command set.



Pre-Testing New Configs

- Can use a similar technique for testing newly developed config files:

```
/usr/sbin/sendmail -v -Am \  
-C newconfig.cf test@address.com
```

- Be sure to test both "internal" and "external" destinations
- Helps ensure the new config is working before replacing `sendmail.cf`
- You will need to do this on the machine where you expect to use the config

Testing a Non-Running Config

When you generate a new `sendmail.cf` file you would obviously like to be able to test that configuration *without* having to put it into production. After all, there could be an error in the config file, and if you replaced your running configuration with something broken you could start bouncing email.

It turns out we can use the Sendmail command line interface in a similar fashion to the example on the previous slide in order to test an alternate configuration file. The trick is the "-C" flag, which allows you to specify the name of your new configuration file (you can specify full path names here if you want). Note that we're also using the "-v" and "-Am" options as before, and you could use "-f" to specify the envelope from address if you wanted. You must specify at least one recipient address.

When you're testing in this fashion it's important that you test thoroughly. So make sure you try a variety of recipient addresses—both inside and outside of your organization—just to be positive that all types of email are being properly routed. Study the SMTP output from the "-v" option carefully to make sure "the right thing" is happening.

The caveat about testing in this manner is that you generally need to run the `sendmail` command line on the machine where you ultimately plan to deploy the new configuration file. After all, the settings in `local-host-names`, `relay-domains`, and your `mailertables` can have a profound effect on Sendmail's behavior.

Exercises

4.1. Dealing with Mail Queues

Exercise Goal – Give you some experience with the Sendmail mail queues and the commands for flushing email out of the queue directories. Since we'll need to stop the MTA process to get mail to queue, let's also practice some alternate mechanisms for stopping and starting the Sendmail processes on a machine.

4.1.1. In order to cause mail to be held in the MSP queue directory, we want to shut down the MTA process on one of the class mail servers. Use the `ps` and `kill` commands to selectively shut down just the MTA process but leave the MSP queue runner process alone. Write down the commands you use in the space below.

Hint: Remember the MTA process is the one running as the `root` user.

4.1.2. Now use the standard Unix command-line mail client to send a piece of email to some username in the `sysiphus.com` domain. Use the appropriate command to print the mail queue and display your queue message. Can you find the files corresponding to your message in the MSP queue directory? Take notes in the space below.

Hint: Don't forget the "`-Ac`" option to specify the MSP queue.

4.1.3. Start the MTA process again, but this time start the process manually from the command line rather than using the `/etc/init.d/sendmail` script. Add the appropriate option to tag the MTA's log entries with the string `"sm-mta"` instead of `"sendmail"`, and also set the queue interval to 4 hours. Write down the complete command line in the space below.

4.1.4. Now flush the MSP queue manually to force any queued messages on their way. Use the verbose option to watch the messages being flushed. Write down the command you used below.

4.2. Diagnosing Bad Email

Exercise Goal – Get some experience interpreting actual mail headers looking for various bits of information that can be helpful in different situations.

4.2.1. In your home directory you should find a file called "testcase1"—this file is an actual piece of spam that I received. Suppose I wanted to block future SMTP connections from the source of this spam. What IP address should I block?

Hint: Look for the first "Received:" header added by a Deer Run mail server.

4.2.2. Look at the file called "testcase2". Apparently this file is a bounce message to me from a remote mail server. But I claim I never sent the original email message and don't even know anybody in the metrogr.org domain. Can you tell what happened here and why I got this bounce message?

Hint: There are two sets of "Received:" headers in this message—the first set isn't very interesting...

4.3. Email Forgery and Email Logs

Exercise Goal – Being able to test your email configuration by injecting email on port 25 of an MTA is a useful technique. Plus this will give us the opportunity to practice our log analysis skills.

4.3.1. Connect to port 25 on one of your mail servers and compose an email message. Set the envelope from to `bogus@deer-run.com` and use some username in the `sysiphus.com` domain as the recipient. Take notes on this process in the space below, and be sure to write down the queue ID number that the remote MTA returns to you when you submit the email message because we're going to use this value in the next exercise.

4.3.2. Now use the queue ID you wrote down in the previous exercise to grab the log messages about your forged email. Can you spot the envelope from and to and the IP address of the machine you initiated the `telnet` connection from? How about the number of recipients and the message ID? Take notes in the space below.

Answers to Exercises

4.1. Dealing with Mail Queues

4.1.1. In order to cause mail to be held in the MSP queue directory, we want to shut down the MTA process on one of the class mail servers. Use the `ps` and `kill` commands to selectively shut down just the MTA process but leave the MSP queue runner process alone. Write down the commands you use in the space below.

Obviously, the process IDs you use will end up being different from the example below, but the process should look something like:

```
# ps -ef | grep send
root  3519      1  0 20:20 ?          00:00:00 sendmail:
accepting connections
smmsp  3525      1  0 20:20 ?          00:00:00 sendmail:
Queue runner@01:00:00 for /var/spool/clientmqueue
root  3588  3573  0 20:54 pts/2    00:00:00 grep send
# kill 3519
```

Remember the MTA process is the root-owned process with PPID 1. The "sendmail: accepting connections" text where the process name should be is also a clue.

If you run the "`ps -ef | grep send`" command line again, you should see only the queue runner process operating.

[Answers to Exercises continue on next page...]

4.1.2. Now use the standard Unix command-line mail client to send a piece of email to some username in the `sysiphus.com` domain. Use the appropriate command to print the mail queue and display your queue message. Can you find the files corresponding to your message in the MSP queue directory? Take notes in the space below.

Here's a quick way to generate a test message:

```
echo test message | mail mary@sysiphus.com
```

The words "test message" become the body of the email message to `mary@sysiphus.com`. Obviously you can specify any message and any recipient (or even a list of recipients) by changing the above command line.

Remember that the command to display the mail queue is either `mailq` or `/usr/sbin/sendmail -bp`. However, we have to add the `-Ac` option to display the MSP queue:

```
# /usr/sbin/sendmail -Ac -bp
      /var/spool/clientmqueue (1 request)
--Q-ID- --Size-- -----Q-Time----- --Sender/Recipient--
k003615      13 Wed Jan  4 21:27 root
      (Deferred: Connection refused by [127.0.0.1])
                                mary@sysiphus.com
      Total requests: 1
```

The output above tells you that the MSP queue directory is `/var/spool/clientmqueue`. You should be able to use the queue ID from the "sendmail -bp" output to locate both a `qf...` and a `df...` file that contain your message.

4.1.3. Start the MTA process again, but this time start the process manually from the command line rather than using the `/etc/init.d/sendmail` script. Add the appropriate option to tag the MTA's log entries with the string "sm-mta" instead of "sendmail", and also set the queue interval to 4 hours. Write down the complete command line in the space below.

The order of options is unimportant, but your command line should look something like:

```
/usr/sbin/sendmail -bd -q4h -L sm-mta
```

`-bd` to tell the MTA to listen on port 25, a queue interval of 4 hours, and `-L` to set the tag for log messages to "sm-mta".

4.1.4. Now flush the MSP queue manually to force any queued messages on their way. Use the verbose option to watch the messages being flushed. Write down the command you used below.

Remember to use the "-Ac" flag to specify the MSP queue, and it's "-v" for verbose mode:

```
# /usr/sbin/sendmail -Ac -q -v
```

```
Running /var/spool/clientmqueue/k0550J0U003615
(sequence 1 of 1)
mary@sysiphus.com... Connecting to [127.0.0.1] via
relay...
220 intl.sysiphus.com ESMTP Sendmail 8.13.4/8.13.4;
Wed, 4 Jan 2006 21:18:05 -0800
>>> EHLO intl.sysiphus.com
250-intl.sysiphus.com Hello localhost.localdomain
[127.0.0.1], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-EXPN
250-VERB
250-8BITMIME
250-SIZE
250-ETRN
250-AUTH DIGEST-MD5 CRAM-MD5
250-DELIVERBY
250 HELP
>>> VERB
250 2.0.0 Verbose mode
>>> MAIL From:<root@sysiphus.com> SIZE=332
AUTH=root@intl.sysiphus.com
250 2.1.0 <root@sysiphus.com>... Sender ok
>>> RCPT To:<mary@sysiphus.com>
>>> DATA
250 2.1.5 <mary@sysiphus.com>... Recipient ok
354 Enter mail, end with "." on a line by itself
>>> .
```

[...much more output not shown...]

If you look carefully at the output, you'll see that the MSP process is turning on the VERB option at the beginning of the SMTP session. This causes the MTA process to display the SMTP session it uses to relay the email on to the next hop mail server (which is the part of the output that I'm not showing you). Very useful.

4.2. Diagnosing Bad Email

4.2.1. In your home directory you should find a file called "testcase1"—this file is an actual piece of spam that I received. Suppose I wanted to block future SMTP connections from the source of this spam. What IP address should I block?

Here are the "Received:" headers from the message:

```
Received: from bullwinkle.deer-run.com (deer.deer-run.com [10.66.1.2])
    by deer.deer-run.com (8.11.7p1+Sun/8.11.6) with ESMTP id ...
    for <info@loopme.deer-run.com>; Wed, 14 Dec 2005 10:18:45 ...
Received: from n082.scl.cp.net (smtpout0157.scl.cp.net [64.97.136.157])
    by bullwinkle.deer-run.com (8.12.11/8.12.11) with ESMTP id ...
    for <info@deer-run.com>; Wed, 14 Dec 2005 10:18:45 -0800 (PST)
Received: from fh1003.dia.cp.net (64.97.139.2) by n082.scl.cp.net
    (7.2.069.1) (authenticated as glory.williams@virgin.net)
    id 439FD7C80001502E; Wed, 14 Dec 2005 17:50:09 +0000
Received: from Http Client 81.69.169.217 by 64.97.168.13 for Recipient(ID
    Suppressed); 2005-12-14 17:50:09 UTC
```

What you want to look for is the IP address of the machine that first relayed the email to a machine in the deer-run.com domain. Counting from the top, take a look at the second "Received:" header above. Notice that header says "Received: ... by bullwinkle.deer-run.com..."? It's the first header that's added by a Deer Run server. So what we want is the IP address in the "from" portion of this header, or 64.97.136.157.

[Answers to Exercises continue on next page...]

4.2.2. Look at the file called "testcase2". Apparently this file is a bounce message to me from a remote mail server. But I claim I never sent the original email message and don't even know anybody in the metrogr.org domain. Can you tell what happened here and why I got this bounce message?

The trick in this case is to look at the "Received:" headers in the original message that was returned by the remote mail server. Here are the relevant lines:

```
Received: from postal (mithril@localhost)
        by securemail.metrogr.org with SMTP id k058Ksb44759
        for <brent@metrogr.org>; Thu, 5 Jan 2006 03:20:54 -0500 ...
        (envelope-from hal@deer-run.com)
Received: from postal.metrogr.org (postal [64.186.50.2]) by
        securemail.metrogr.org ([192.168.132.31]); 05 Jan 2006 03:20:54 ...
Received: from deer-run.com (219.64.185.5.ahd.dialup.vsnl.net.in
        [219.64.185.5]) by postal.metrogr.org; Thu, 05 Jan 2006 03:20:15 -0500
```

Notice the "Received:" header at the bottom? The initial sending host claimed to be "deer-run.com", but the actual DNS resolution of the machine was some random host in the "dialup.vsnl.net.in" domain. Also in the top "Received:" header you can see the mail server at metrogr.org identifying the envelope from as hal@deer-run.com.

This message is what is usually referred to as "backscatter". Some machine, probably infected with a Windows worm or virus, is sending out virus-laden emails to random email addresses. The application sending the email is smart enough to try and hide itself by choosing random email addresses as the envelope from address. Unfortunately, when the recipient of the email message doesn't actually exist, the bounce message is returned to the sender as specified by the envelope from. So I really never did communicate with the metrogr.org domain, I just got hit with the backwash.

[Answers to Exercises continue on next page...]

4.3. Email Forgery and Email Logs

4.3.1. Connect to port 25 on one of your mail servers and compose an email message. Set the envelope from to `bogus@deer-run.com` and use some username in the `sysiphus.com` domain as the recipient. Take notes on this process in the space below, and be sure to write down the queue ID number that the remote MTA returns to you when you submit the email message because we're going to use this value in the next exercise.

Here's a typical SMTP session:

```
# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
220 int1.sysiphus.com ESMTP Sendmail 8.13.4/8.13.4; ...
helo localhost.localdomain
250 int1.sysiphus.com Hello localhost.localdomain ...
mail from: bogus@deer-run.com
250 2.1.0 bogus@deer-run.com... Sender ok
rcpt to: hal@sysiphus.com
250 2.1.5 hal@sysiphus.com... Recipient ok
data
354 Enter mail, end with "." on a line by itself
From: youcouldbeawinner@spam.com
To: unreal@some.com
Subject: testing testing

1 2 3
.
250 2.0.0 k056SDUq003739 Message accepted for delivery
quit
221 2.0.0 int1.sysiphus.com closing connection
Connection closed by foreign host.
```

See the line that reads, "k056SDUq003739 Message accepted for delivery"? That tells us the queue ID value we're going to need to look up the log entry on the remote MTA.

4.3.2. Now use the queue ID you wrote down in the previous exercise to grab the log messages about your forged email. Can you spot the envelope from and to and the IP address of the machine you initiated the `telnet` connection from? How about the number of recipients and the message ID? Take notes in the space below.

Here are the log entries from my test case:

```
# grep k056SDUq003739 /var/log/maillog
Jan  4 22:29:53 int1 sm-mta[3739]: k056SDUq003739:
from=bogus@deer-run.com, size=85, class=0, nrcpts=1,
msgid=<200601050628.k056SDUq003739@int1.sysiphus.com>,
proto=SMTP, daemon=MTA, relay=localhost.localdomain
[127.0.0.1]
Jan  4 22:30:04 int1 sm-mta[3741]: k056SDUq003739:
to=hal@sysiphus.com, delay=00:01:15, xdelay=00:00:11,
mailer=smtp, pri=120085, relay=exchange.sysiphus.com.
[192.168.1.1], dsn=2.0.0, stat=Sent (k05L67o01165
Message accepted for delivery)
```

In the first log entry you can see the envelope from, the number of recipients, the message ID, and the host that sent us the email all in bold face. On the second line you can see the envelope recipient address.

Performance Tuning



Performance Tuning

Most sites never have to worry about Sendmail performance tuning, because even the default configuration of Sendmail can handle tens of thousands of messages per day. However, after completing this module you should:

- Understand how to deploy multiple relay servers in parallel to achieve greater message throughput
- Understand the normal file system bottlenecks for Sendmail and how to improve performance
- Understand queue bottlenecks and how to resolve them
- Know where and how to deploy dedicated DNS servers to improve Sendmail performance
- Understand the internal Sendmail configuration variables that can affect performance



Caution...

- The first rule of Sendmail performance tuning is *don't do it!*
- Wait until you actually have a problem before fiddling with things
- Modern hardware can deliver huge amounts of email with default config

Don't Tune If You Don't Have To

Every talk on Sendmail performance tuning begins with the same piece of advice: *DON'T*. For the vast majority of installations, Sendmail running on modern hardware can handle the required volume of mail—tens of thousands of email messages per day at least. It doesn't take that much *oomph* to relay email.

Plenty of sites have wasted a lot of time and money tuning the performance of their mail servers only to discover they've been tackling the wrong issues. Or they allow their mail servers to operate so quickly that they're able to saturate their Internet connection, throttling not only their mail service, but also all of their other Internet traffic as well. Or they pass email so quickly that it overwhelms their internal Windows-based email servers.

So the moral of the story is, wait until you actually have a problem with getting mail delivered before you start tweaking Sendmail. That way you'll at least have data on where the bottlenecks are actually occurring, rather than speculating on where they might occur. And maybe you'll never have to bother tuning Sendmail at all.



Parallelization

- When used as a relay, Sendmail "parallelizes" very easily
- Having a small farm of inexpensive machines is often the best solution
- Simple DNS "round robin" is usually enough to spread the load:

```
sysiphus.com.  IN  MX 10  ext1.sysiphus.com.  
sysiphus.com.  IN  MX 10  ext2.sysiphus.com.  
sysiphus.com.  IN  MX 10  ext3.sysiphus.com.
```

Increasing Throughput Through Parallelization

Rather than spending a lot of time hacking on a single mail server trying to get it to process email as quickly as possible, often the best method to increase your overall email throughput is to simply add more servers. Given how cheap hardware is these days, it's quite inexpensive to create a small farm of Linux machines running on commodity hardware to act as mail relays. Besides, that way you have "n+1 redundancy"—you can lose one of your relay servers and the other machines that are still functioning can pick up the slack.

Generally speaking, parallel server configurations only work when the mail server is acting as a relay. If the server is actually storing user mailboxes, it's not easy to have multiple mail servers "sharing" a single user's mailbox. Really big ISPs (AOL, etc) have to do this kind of thing of course, but this kind of architecture is *way* beyond what I intend to show you in this course.

Suppose we decided to replace our single external mail relay with three servers—call them `ext1.sysiphus.com`, `ext2.sysiphus.com`, and `ext3.sysiphus.com`. The question is how do we get the outside world to spread our incoming email volume across these three machines? Turns out that you're allowed to have multiple MX records at the same priority level as you see in the example on the slide. What happens in this case is that the first host that asks for the MX record for `sysiphus.com` gets the list of all three MX servers with `ext1.sysiphus.com` listed first—that host will try to send the email to `ext1` first and then try the other servers only if `ext1` is down. The next server to request the MX for `sysiphus.com` will get the servers listed with `ext2` appearing first in the list. The next server gets a list with `ext3` at the top of the list and

so on. This is called "DNS round-robin" and is usually sufficient for spreading the load out among your mail servers.

Note that you can set up exactly the same configuration for your internal relays as well. So far in all of our examples we've been using the name "internal.sysiphus.com" in places like the MAIL_HUB directive on the external servers. There's no reason that name can't be associated with a group of MX records instead of an A record:

```
internal.sysiphus.com.  IN  MX 10  int1.sysiphus.com.  
internal.sysiphus.com.  IN  MX 10  int2.sysiphus.com.  
internal.sysiphus.com.  IN  MX 10  int3.sysiphus.com.
```

Now whenever one of your machines wants to forward a piece of email to internal.sysiphus.com, they'll end up falling into the same DNS "round-robin" pattern described above.

The only problem with the basic DNS round-robin is that your DNS information gets cached at remote sites. You probably get a lot of email from AOL and other large ISPs. Well AOL is going to request the MX records for your domain *once* and then hold onto that information for some period of time—this is referred to as the "time to live" setting for your DNS domain. During the "time to live" period, all of the email from AOL is going to go through whatever MX server was at the top of the DNS round-robin list at the time AOL made their query, and that external relay is going to see relatively more traffic than your other servers. Usually this isn't such a big deal, but if it becomes a real problem you can smooth the load out by artificially reducing the "time to live" value to force remote sites to re-query the DNS round-robin more frequently (at the cost of putting more load on your DNS servers with all of the extra query activity).

For BIND running on Unix, you can actually specify the time to live value on a per-record basis:

```
sysiphus.com. 15m IN MX 10 ext1.sysiphus.com.  
sysiphus.com. 15m IN MX 10 ext2.sysiphus.com.  
sysiphus.com. 15m IN MX 10 ext3.sysiphus.com.
```

See the "15m" entries we inserted after the domain name? That's specifying a time to live value of 15 minutes for each record. You could even make the time window shorter if you wanted.



Typical Bottlenecks

- CPU power is rarely an issue, although more CPUs help concurrency
- Sendmail has large memory footprint, but modern shared memory helps
- Network issues:
 - Upstream network bandwidth
 - DNS queries
- But mostly the problem is disk I/O:
 - Synchronous writes to mail queues
 - "Deep" mail queues

Typical Sendmail Performance Bottlenecks

Sendmail itself is rarely ever CPU-bound, because SMTP just does not take that much CPU power. Yeah, if you had a multi-CPU machine you could run more Sendmail child processes in parallel, but I've actually heard of sites that regularly have over a hundred concurrent Sendmail processes all sharing a single CPU. CPU power becomes a factor when you start doing heavy virus scanning (and anti-spam stuff to a lesser degree), but for a simple relay CPU is not a big deal.

Sendmail has a somewhat deserved reputation for being a large, bloated binary. And so you'd worry about having lots of concurrent Sendmail processes sucking up all of your memory. But the reality is that modern Unix architectures are very memory efficient because they "share" the same binary image across all of the different Sendmail processes that are running. So throw a couple of gigs of RAM into each of your mail relays and forget about memory as an issue.

Network performance can be an issue, but if you have fast systems with good network cards, you can probably saturate your upstream Internet connection long before throughput becomes an issue for Sendmail. One networking issue that can become a problem for high-volume mail relays is DNS, because Sendmail makes a lot of DNS queries. We'll tackle this problem a bit later in this section.

The first Sendmail bottleneck that almost every site hits is file I/O. There are a couple of different issues that you run into with Sendmail in this area, and we'll talk about each of them in the next few slides.



Synchronous Write Problem

- When receiving email, Sendmail must write and flush two queue files to disk
- This destroys file system performance on normal file systems
- Standard work-arounds:
 - Use a journaling file system (VxFS, XFS)
 - NVRAM write cache (good RAID hardware)
 - Expensive silicon disks

Synchronous Writes

Sendmail goes to extreme lengths to never lose email. When your MTA is receiving a message from another machine, it insists on writing the incoming message into the `mqueue` directory and *flushing the queue files to disk* before it's willing to confirm receipt of the message. Modern file systems are optimized to buffer write operations like disk writes and re-order them in a more efficient fashion. When Sendmail forces the operating system to flush its queue files to disk all the time, it defeats the optimization algorithms in the file system and usually wreaks havoc with your file system performance.

The easiest optimization is to use faster disks. Higher RPMs is good, but more important is the type of disk controller you're using. Avoid IDE and go for fast-wide ultra-SCSI if you can afford it. I haven't actually tried running a mail server using the new faster SATA technology, but all of the performance benchmarks I've seen still put ultra-SCSI ahead of SATA. It actually turns out that many small disk drives (RAID) is faster than a single large drive—not only because you can spread the writes out across multiple disks, but also because the seek times tend to be shorter on small drives. Unfortunately, the trend is for big disk drives. I've actually seen sites that are forced to get big drives (100GB or more), so they partition the drives so that they're only using the first 16GB or so of the drive and "wasting" the rest of the space, just to reduce their average seek times.

When the raw speed of the disk drive isn't sufficient, then you start playing games. Your choice of file system can make a difference. "Journaling" file systems (Veritas VxFS or SGI's XFS, which is also available on Linux) stick write operations into a "journal" or transaction log, kind of like a "two-phase commit" operation used by modern relational databases. Sophisticated journaling file systems can then re-order write operations in the

journal to improve disk performance. Because messages in the `mqueue` directory are usually only there for a very small amount of time (most email messages get delivered in less than two seconds), it's actually possible for Sendmail to deliver the message and request that the queue files be deleted before the original request to create the files that's sitting in the journal ever gets acted upon. When this happens, the actual file write never occurs—the journaling file system recognizes that the combination of writing the file and deleting the file cancel each other out and no change is actually made to the file system.

You can also see the same "canceling" behavior with high-quality RAID hardware. Expensive RAID hardware improves performance by having a non-volatile memory cache (NVRAM) that buffers up write operations to the RAID array similar to the journal in a journaling file system. But since the NVRAM is memory-based rather than disk-based it's even faster. And if you really want to spend some serious money you can invest in "silicon disk" technology, where the entire storage system uses NVRAM rather than traditional disk-based storage.



"Deep" Queues: Problem #1

- Messages to downed sites and spam bounces collect in the queue
- At each queue interval, a new process is forked to read/process entire queue
- If queue processing takes longer than queue interval, problems occur:
 - Concurrent queue runner processes start having lock contention issues
 - More and more queue runners build up until system locks up under the load
- One solution is to shut off Sendmail and move stalled email to alternate queue

Problems with "Deep" Queues

Queue Runner Contention

It's a fact of life that messages will collect in your mail queues. The destination site may be temporarily down, or never have been up in the first place (recall Bob's auto-responder messages that were trying to go back to a spammer's mail servers). All of these undeliverable messages can really bog down your queue runner processes.

Where you start having real problems is when you've got so many blocked messages that it takes longer than your queue interval ("`-q1h`") to process the queue. This means that a second queue runner process is going to start before the previous one finishes. The reason this is a difficulty is that you get into "lock contention" issues where the two queue runner processes end up blocking each other as each waits for the other queue runner to lock and unlock the various queue files. In other words, two queue runners make the queue processing go even slower. Which means maybe a third queue runner gets started an hour later, further bogging down the process. In the worst case scenario, none of the queue runners ever gets a chance to finish—you just keep spawning more and more queue runners until the system chokes and dies.

At the end of this section, I'll show you some Sendmail options to limit the number of simultaneous queue runners. However, all of those blocked messages can still delay the delivery of queued messages that actually could be delivered. Some sites will actually move "stuck" messages (messages that are say more than a day old) to another queue directory.

Let's say you've already created a directory called `/var/spool/slow-mqueue`. Every few hours you could run the following shell script from cron:

```
#!/bin/sh

/etc/init.d/sendmail stop
find /var/spool/mqueue -type f -mtime +1 \
    -exec mv {} /var/spool/slow-mqueue \;
/etc/init.d/sendmail start
```

First we shut down the Sendmail processes. Then we run a `find` command to locate all files in the `mqueue` directory that are more than 1 day old ("`-mtime +1`") and move them into the `slow-mqueue` directory. Then we restart the Sendmail daemons.

Now you're going to have to arrange for some Sendmail process to actually try to flush the `slow-mqueue` directory every so often. One way to do this is via cron:

```
0 0,4,8,12,16,20 * * * /usr/sbin/sendmail -q \
-O QueueDir=/var/spool/slow-mqueue >/dev/null 2>&1
```

This cron job will run every four hours and invoke "`sendmail -q`" to flush the `slow-mqueue` directory (we're using the "`-O QueueDir=...`" command line syntax to specify the alternate queue directory instead of the standard `/var/spool/mqueue`). Another alternative would be to start a queue runner for the `slow-mqueue` directory as a separate process:

```
/usr/sbin/sendmail -q4h \
-O QueueDir=/var/spool/slow-mqueue
```

You'd need to adjust your boot scripts to make sure this process got started automatically by your `/etc/init.d/sendmail` script.



"Deep" Queues: Problem #2

- When there are lots of queued messages, directory gets "large"
- Standard Unix file system has poor performance on large directories
- Solutions:
 - Use alternate file system (VxFS, XFS)
 - Split up queue directories (also split across multiple disks for better I/O performance)
 - Regularly move stalled email to alternate queues and create new queue dirs

Flabby Queue Directories

The sheer number of files in a queue directory can cause you problems as well. The standard Unix file system implements directories as a simple list of file names. When the number of files in the directory becomes large, doing linear searches through the directory to find the file you want starts taking longer and longer. And the real trouble is that once your directory gets "big" it doesn't shrink again even after you clean out all the stuck files, so performance still suffers.

Maybe the easiest solution is to use a file system that doesn't treat directories as big long lists of files, but instead makes directories into simple databases so that file lookups are fast. Candidate file systems include Veritas' VxFS, SGI's XFS, and even ReiserFS under Linux.

Sendmail also allows you to split up your queue into multiple directories. The simplest way to do this is to create directories called "qf" and "df" under `/var/spool/mqueue`. Sendmail will then automatically split queue files out into the appropriate directory. Actually, there's another type of queue file that I didn't tell you about—"xf" files. Sendmail uses xf files to temporarily store error messages from different mail servers when it's in the middle of sending an email to multiple recipients. These xf files are very short-lived because they're deleted as soon as the delivery attempt is completed. Still, you should create a `/var/spool/mqueue/xf` directory in addition to your ".../qf" and ".../df" directories.

You actually are allowed to have several completely different queue directories (possibly each with its own qf, df, and xf subdirectories), but this configuration is beyond the

scope of this course—consult the Performance Tuning chapter (Chapter 6) in the O'Reilly *Sendmail* book for further information. Also, if you're really interested in maximizing performance, you can always split your queue directories out across multiple disk drives.

Some sites will even periodically shut down Sendmail and recreate their queue directories to prevent them from becoming "flabby" and remaining that way. Let's modify the shell script I showed you in the notes section of the previous slide:

```
#!/bin/sh

/etc/init.d/sendmail stop
find /var/spool/mqueue -type f -mtime +1 \
    -exec mv {} /var/spool/slow-mqueue \;
mkdir -m 700 -p /var/spool/mqueue-new
chown sendmail:sendmail /var/spool/mqueue-new
mv /var/spool/mqueue/* /var/spool/mqueue-new
rmdir /var/spool/mqueue
mv /var/spool/mqueue-new /var/spool/mqueue
/etc/init.d/sendmail start
```

Now in addition to moving day old files to the `slow-mqueue` directory, we're creating a new queue directory and moving the contents of `/var/spool/mqueue` into this newly created directory. Then we remove the old and possibly flabby `mqueue` directory and replace it with the `mqueue-new` directory we just created. Note that the above script assumes we're using `RUN_AS_USER` and running the child MTA processes as user `sendmail`. Change the `chown` command in the middle of the script appropriately if this is not the case at your site.

- Also don't allow your other hosts that aren't mail relays to query these servers. The point is to dedicate these machines to supporting your mail infrastructure and not bog them down with normal user DNS activity.



Tuning Sendmail Timeouts

- Already mentioned this one:

```
define(`confTO_IDENT', `0s')
```

- Reduce amount of "extra" email:

```
define(`confTO_QUEUEWARN', `4h')
```

- Limit queue retention:

```
define(`confTO_QUEUERETURN', `5d')
```

- Can also limit retry frequency:

```
define(`confMIN_QUEUE_AGE', `2h')
```

- Tuning internal SMTP timeouts is also possible, but beyond this course level

Tuning Timeout Values

Sendmail has a number of internal timeouts that you can fiddle with to possibly improve performance. We already talked about setting `TO_IDENT` to prevent Sendmail from hanging on pointless Ident queries.

Other interesting parameters include `TO_QUEUEWARN`, `TO_QUEUERETURN`, and `MIN_QUEUE_AGE`:

- By default, Sendmail will send a warning message back to the message sender if a given email message gets stuck in the queue for more than 4 hours. However, you can tune this value by setting `TO_QUEUEWARN`. Maybe you want to increase this limit to 8 hours to reduce the amount of extraneous warning messages you send out. Most messages get sent within a few seconds anyway, and messages that get stuck in the queue tend to get stuck for a long time, so tuning this value isn't a huge deal.
- `TO_QUEUERETURN` is the number of days a message will be kept in the queue before Sendmail gives up and kicks the message back to the sender with an error. The default here is 5 days, but you might want to shorten this interval to throw messages out of the queue a bit faster. But making this value too short could cause you to bounce legitimate email if a site has a major meltdown that takes more than a couple of days to fix. You're probably better off moving stuck messages to an alternate message queue as we discussed previously.
- You also may not want to try and deliver every message in the queue on each queue runner pass. Messages that are stuck tend to be stuck for a while. `MIN_QUEUE_AGE`

is the minimum amount of time since the last delivery attempt that must pass before the queue runner will attempt delivery. You might increase this to a value like 4 hours, but if you do that you'll definitely want to change the setting on `TO_QUEUEWARN` as well. Make sure you make at least a couple of attempts to deliver each message before you hit the `TO_QUEUEWARN` limit.

Sendmail has a bunch of other timeout values that control every phase of the SMTP communication—how long you wait for the remote site to put up its greeting string, how long you wait for the remote site to acknowledge your `HELO`, etc. For the most part these timeouts are set *very* conservatively, and this can result in long delays on hosts that are down or overloaded. Chapter 6 in the O'Reilly *Sendmail* book has some ideas on tuning these parameters to improve performance, but this configuration is beyond the scope of this course.



Sendmail Throttling

- The defaults may be too low here, and also may be backwards:

```
define(`confQUEUE_LA', `8')
define(`confREFUSE_LA', `12')
```

- Helps to flatten out "bursts" of email:

```
define(`confCONNECTION_RATE_THROTTLE', `5')
```

- Put limits on forking:

```
define(`confMAX_QUEUE_CHILDREN', `2')
define(`confMAX_DAEMON_CHILDREN', `24')
```

Throttling Controls

The worst thing that can happen is when a process runs amok and consumes so many resources that the system becomes unusable. You want systems to "degrade gracefully" as traffic mounts up. Sendmail has a number of built-in controls in this area, and you can tune various parameters to make Sendmail behave more appropriately for the particular hardware it's running on.

QUEUE_LA represents the system load average* above which Sendmail stops trying to immediately deliver new email messages and starts simply dropping these messages in the queue for later delivery once things have calmed down. REFUSE_LA is the system load average above which Sendmail starts refusing to accept new SMTP connections from external servers. Unless you have a very busy server, these values rarely become a factor, but it's possible that the default values are set too low for modern hardware. It's not uncommon to see busy Sendmail relays operating at load averages in the 50-100 range, and there have been reports of mail relays operating with consistent load averages above 200.

More interesting is the argument that REFUSE_LA should actually be set *lower* than QUEUE_LA. After all, if your mail server is really overloaded, you're not doing yourself any favors by accepting a bunch of email and stuffing it into the queue where future delivery attempts will eat up even more system resources. Maybe you want to start refusing new SMTP connections until you get the current situation under control.

* You can think of the system load average as being the number of processes that are simultaneously fighting for CPU time on the machine.

`CONNECTION_RATE_THROTTLE` is a useful mechanism for spreading out "bursts" of incoming SMTP connections that all happen in a short interval. In the example on the slide, we've set this parameter to 5. Now suppose 20 new SMTP connections all come in within the same one-second interval. With our `CONNECTION_RATE_THROTTLE` setting, the first five connections will be handled in the first second, then the next five one second later, then the next five, and so on. This really helps to even out your load and can even slow down spammers who are hammering your mail servers.

Earlier we talked about the problem where so many queue runner processes end up getting created that your system bogs down and becomes unusable. You can set `MAX_QUEUE_CHILDREN` to put a hard limit on the number of simultaneous queue runner processes that may be operating at any given time. `MAX_DAEMON_CHILDREN` puts a similar limit on the number of MTA child processes that can be running and handling new SMTP connections. While `MAX_QUEUE_CHILDREN` can be quite useful in my opinion, I'm a little more leery of `MAX_DAEMON_CHILDREN`. I prefer to let other mechanisms like `CONNECTION_RATE_THROTTLE` control how many SMTP connections I'm handling at any given instant.

Exercises

5.1. Performance Tweaks and Hacks

Exercise Goal – Get some experience creating and using multiple queue directories. Practice setting some of the performance-related parameters in your configuration files.

Before starting this exercise, remind the instructor to break the `sysiphus.com` DNS configuration so that mail will queue up on the test servers we're using in class. It would also be a good idea to remind the instructor to fix it once the exercise is over.

5.1.1. Shut down the Sendmail processes on the mail server. Now add `qf`, `df`, and `xf` subdirectories under the MTA queue directory. Make sure you properly set the ownerships and permissions on these directories. Write down the commands you use in the space below.

Hint: The owner of the new queue directories will vary depending on whether you have `RUN_AS_USER` set or not.

5.1.2. Now modify the configuration for this server. Set `REFUSE_LA` to 20 and `QUEUE_LA` to 40. Also set the connection rate throttle to 10 simultaneous connections. Limit the number of queue runners to 2. Write down the macro configuration directives you use in the space below. Try running `diff` to see the differences between the new configuration and the old one. Overwrite the old configuration with the new one and restart the Sendmail daemons.

5.1.3. Send a test email to some user in the `sysiphus.com` domain. Now print the contents of the mail queue on your machine and see if the message is still in the queue. Go to the MTA queue directory and verify that the queue files have been put in different directories. Take notes on this process in the space below.

Answers to Exercises

5.1. Performance Tweaks and Hacks

5.1.1. Shut down the Sendmail processes on the mail server. Now add `qf`, `df`, and `xf` subdirectories under the MTA queue directory. Make sure you properly set the ownerships and permissions on these directories. Write down the commands you use in the space below.

Here's a sample session—notice that just for the sake of variety I'm using the `"pkill sendmail"` command to stop the Sendmail processes on the machine, rather than using the `/etc/init.d/sendmail` script:

```
# pkill sendmail
# cd /var/spool/mqueue
# mkdir qf df xf
# chown sendmail:sendmail *
# chmod 700 *
```

It would appear that the machine I'm working on is using the `RUN_AS_USER` option and running the Sendmail MTA as the unprivileged `"sendmail"` user and group. If you were doing this process on a machine that didn't have `RUN_AS_USER` set, then the second to last command would be `"chown root:root *" instead.`

5.1.2. Now modify the configuration for this server. Set `REFUSE_LA` to 20 and `QUEUE_LA` to 40. Also set the connection rate throttle to 10 simultaneous connections. Limit the number of queue runners to 2. Write down the macro configuration directives you use in the space below. Try running `diff` to see the differences between the new configuration and the old one. Overwrite the old configuration with the new one and restart the Sendmail daemons.

Here are the appropriate Sendmail macros:

```
define(`confREFUSE_LA', `20')
define(`confQUEUE_LA', `40')
define(`confCONNECTION_RATE_THROTTLE', `10')
define(`confMAX_QUEUE_CHILDREN', `2')
```

[answer continues on next page...]

Here's me building the new .cf file and running the diff:

```
$ m4 internal.mc >internal.cf
*** WARNING: FEATURE(`promiscuous_relay') configures
    your system as open relay.  Do NOT use it on a
    server that is connected to the Internet!
$ diff internal.cf /etc/mail/sendmail.cf
    [...uninteresting lines not shown...]
294c288
< O MaxQueueChildren=2
---
> #O MaxQueueChildren
395c389
< O QueueLA=40
---
> #O QueueLA=8
398c392
< O RefuseLA=20
---
> #O RefuseLA=12
410c404
< O ConnectionRateThrottle=10
---
> #O ConnectionRateThrottle=0
```

And just in case you need to see it again, here's me updating the sendmail.cf file and restarting the mail server:

```
$ /bin/su
Password: <not echoed>
# cp internal.cf /etc/mail/sendmail.cf
# /etc/init.d/sendmail start
Starting sendmail:           [ OK ]
Starting sm-client:         [ OK ]
```


5.1.3. Send a test email to some user in the `sysiphus.com` domain. Now print the contents of the mail queue on your machine and see if the message is still in the queue. Go to the MTA queue directory and verify that the queue files have been put in different directories.

Here's the mail queue after I ran this test:

```
# /usr/sbin/sendmail -bp
      /var/spool/mqueue/df (1 request)
---Q-ID-- --Size-- -----Q-Time---- -Sender/Recipient-
k7pSQ3959      13 Wed Jan 4 23:51 <root@sysiphus.com>
(Deferred: nonexistent.sysiphus.com.: No route to ...)
                                         <hal@sysiphus.com>
      Total requests: 1
```

Notice that on the first line of output Sendmail has listed the queue directory name as `/var/spool/mqueue/df`. I'm actually not sure why it chooses to display the `".../df"` directory name here—after all, the "interesting" file (at least from the queueing perspective) is the file in the `".../qf"` directory.

When I checked the `/var/spool/mqueue` directory, here's what I saw:

```
# ls /var/spool/mqueue/*
/var/spool/mqueue/df:
dfk7pSQ3959

/var/spool/mqueue/qf:
qfk7pSQ3959

/var/spool/mqueue/xf:
```

So the split queue directories seem to be operating as expected.

Spam and Virus Control



Spam and Virus Control

Unsolicited commercial email (aka "spam") and email-borne viruses are obviously huge problems on the Internet today. After completing this module you should:

- Understand the issues you need to consider *before* designing and deploying your anti-spam and anti-virus response
- Be familiar with the built-in anti-spam technologies in Sendmail and how to configure them
- Have been exposed to the Sendmail Milter interface and some of the anti-spam and anti-virus tools that are available as Sendmail Milters
- Understand more advanced spam-fighting techniques like content signature databases and greylisting



The Arms Race

- Spammers are adaptive, changing their tactics as new controls developed
- One measure of effectiveness is an overall reduction in received spam
- Another measure is if spammers are forced to change their tactics
- Ultimately, the battle turns on economics and not clever technology

Before You Begin

The Arms Race

I date the beginning of the war on spam as being about 1994—this was the year that Sendmail v8.9 was released. Sendmail v8.9 is significant here because this was the release that changed many of Sendmail's default behaviors (breaking many sites' mail architectures) in order to make the Internet a less friendly place to spammers. For example, prior to Sendmail v8.9 all Sendmail installations were promiscuous relays by default.

In the time since then we've seen both the volume and the sophistication of spam increase dramatically. Spammers have also changed the way they send spam. In the beginning, spammers used all of the promiscuous relays on the Internet to forward spam wherever they wanted. Once Sendmail's default behavior changed, the spammers were forced to set up their own mail servers to send spam, which was in turn thwarted by "blacklisting" the spammers' mail servers and not accepting SMTP connections from those IPs. Today you see spammers resorting to using Trojan horse programs to infect vulnerable Internet systems and using those systems as "zombies" for transmitting their spam (these zombie networks can also unfortunately be used for distributed denial-of-service attacks). Eventually the zombies get shut down and/or cleaned up by standard anti-virus and anti-spyware tools, forcing the spammers to go and find more machines to infect.

Certainly the anti-spam metric that most sites care about is the amount of spam they're able to stop before it gets into users' mailboxes. But I don't accept the notion that spammers have the right to hammer my mail servers with spam and it's my job to block

as much of it as I can. I want to know that we're making progress in stopping spamming behavior entirely.

To me it seems that the issue is economic, not technological. Until sending spam costs the spammer more than the compensation received for sending the spam in the first place we will continue to have spam. If you want to know if we're "winning" the war on spam, see if we're successful in making the spammers spend more money to send their spam. I think we're slowly doing this. After all, having to deploy their own mail servers was more expensive for the spammers than just using open relays scattered all over the Internet. And developing technology to infect machines and make zombies is even more costly—and ultimately opens the spammers up for criminal prosecution.

But it's a long, hard process. And in the meantime there is a lot of spam to stop. Plus spammers are always changing their tactics, so you always have to be adjusting your anti-spam filters to keep up.



Stop and Think!

- Aggressive anti-spam controls increase your risk of false-positives
- Spam control is a business decision
- Discuss your plans with senior management *prior* to implementation
- Consider "tagging" spam rather than blocking it completely

Appropriate Levels of Spam Control

One way to stop spam from reaching your users is to simply block all inbound email to your organization. Of course you'd be blocking legitimate email as well, so you might not have your job for long, but for one bright shining moment you wouldn't be receiving any spam.

All kidding aside, as you start ramping up your anti-spam responses the problem of "false positives"—mistakenly rejecting legitimate email—becomes a real issue. This is definitely a business decision. You need to educate your management on the options available to the organization and the potential costs, and then let them make the call on the appropriate level of spam control for your organization. If you don't do this, you tend to end up on the wrong side of conversations with the VP of Customer Service explaining why you've been telling your company's customers that they're spammers and you won't accept email from them.

These days most organizations seem to be migrating toward solutions where possible spam is "tagged" (usually with special headers inserted into each message). Your users' mail clients can then filter incoming email based on these tags and automatically move messages tagged as spam into a separate folder where the user never has to look at it. That way, if there is a false positive your users can go into their spam folder and retrieve the legitimate email message—it hasn't been totally eradicated.

The obvious downside to this strategy is that you still have to expend the resources to receive the spam, tag it, and then forward the tagged spam to your users. Plus there's the cost of filtering and storing the tagged spam. So what you'd like is a system that abolishes the email that you're 100% sure is spam and then tags and forwards the rest.



Hard Truths

- Spam volume is increasing dramatically
- Virus scanning adds significant system load on top of spam control tools
- High volumes + heavy processing means more expensive hardware
- Managing and tuning spam and virus controls requires significant resources
- Outsourcing is becoming popular...

Increasing Costs

Spam volume continues to rise—a trend which shows no real signs of slowing. Automated email viruses are also dramatically increasing the volume of email your external relays have to deal with. On the other side of the coin, anti-spam and particularly anti-virus scanners have to become increasingly sophisticated to deal with new spam and new attack vectors. This means more processing power is required to analyze each message.

Higher message volumes plus increasing computational complexity per message means that you have to constantly upgrade your infrastructure to deal with the load. It's like somebody telling you that you need to buy three new fax machines just to process junk faxes. Insanity. And of course there are the administrative costs associated with upgrades, new hardware installations, and deploying new anti-spam and anti-virus techniques to keep up with the "arms race".

I believe that it is possible for an organization to deploy technology that effectively blocks spam headed towards their users (I do a pretty good job of this for Deer Run and my other customers). But it's a full-time job and organizations may lack the in-house expertise to handle this. So it's not surprising that outsourcing is becoming popular. Outsourcing spam and virus filtering makes a lot of sense to me—not only can the outsourcing provider achieve greater economies of scale than a single organization, but also the more spam they filter the more exacting their spam filters become.

Outsourcing takes many forms. You can literally force all of your inbound and outbound mail through somebody else's managed servers—this is the MessageLabs model. Or you can "subscribe" your own mail servers to a database of spam signatures (sort of like an

anti-virus product, but this one recognizes spam), which is Brightmail's architecture. Or you can buy an "appliance" type solution that you just plug into your network—usually in place of your external relay servers. There are too many anti-spam appliances on the market to list at this point.



Where to Deploy

- Spam control has to happen on your external mail relays
- Virus control can happen "anywhere" as long as all email gets scanned
- Open Source solutions generally combine anti-spam with anti-virus
- May ultimately have to deploy dedicated virus scanning cluster

Deployment Considerations

Anti-spam controls need to be deployed on your external mail relays. This is because the IP address of the system making the SMTP connection to your server is usually one of the inputs that helps you decide whether or not the message is spam. Also, deploying your anti-spam controls at your external relays gives you the option of dropping certain traffic before it even gets into your organization.

There are more options when it comes to locating your anti-virus response. Historically, organizations were primarily interested in stopping email-borne viruses coming in from outside the company and so anti-virus products were located on or near the external gateways. Also, the popular Open Source tools in this area—MIMEDefang, for example—tend to couple anti-spam and anti-virus together on the same machine.

Now, however, there is significant concern about virus outbreaks within an organization and stopping virus-ridden email from migrating around the internal infrastructure. So anti-virus is moving "inside". It doesn't really matter where you locate your anti-virus response, just so long as you ensure that all of your email (internal and external) gets scanned.

Actually this flexibility is something of a blessing in disguise. Large organizations are finding that they need to deploy clusters of virus scanners to handle their entire email volume. So it's nice to be able to decouple the virus-scanning piece onto its own set of servers and just push inbound and outbound mail through this infrastructure.



Default Sendmail Tools

- Promiscuous relaying not allowed
- Reject mail from unqualified senders
- Reject mail from invalid domains
- Optional functionality can be enabled:
 - "GreetPause" option to stop "slammers"
 - Access database for manual blacklists
 - Automated DNS-based blacklists (`dnsbl`)

Sendmail's Built-In Anti-Spam Features

Default Behaviors

Starting with Sendmail v8.9, Sendmail does not allow promiscuous relaying. We've already discussed how the `local-host-names` and `relay-domains` files help Sendmail detect which emails are legitimate and how to use `FEATURE(`promiscuous_relay')` to disable these checks on mail servers that are not accessible from the Internet.

Sendmail v8.9 also introduced controls on unqualified sender addresses—recall our discussion of the "accept_unqualified_senders" feature and masquerading. This version of Sendmail also started doing sender domain validation, refusing to accept email if the sender address was for a domain that didn't exist. Like the other anti-spam checks in Sendmail, you can disable this behavior with `FEATURE(`accept_unresolvable_domains')`, but it's unclear why you would ever want to do this—unless perhaps on a machine that was unable to make DNS queries because it was behind a firewall.

Aside from these basic default checks, Sendmail also includes several pieces of optional functionality that you can enable. This includes the `GreetPause` timeout for detecting fraudulent mailers, the access DB for manually maintaining "blacklists" of sites you don't wish to communicate with, and the "dnsbl" feature for subscribing to blacklists maintained by other organizations. We'll discuss all of these in some detail in the next several slides.



GreetPause

- Remote end is supposed to wait for your SMTP greeting before sending
- "Slammers" just fire entire SMTP session at you without pausing
- Legitimate senders should not do this
- "GreetPause" defines wait time for greeting, rejects "slamming" attempts
- Creates latency on each new email, so keep timeout value low:

```
FEATURE(`greet_pause', `1000')
```

The GreetPause Feature

When a remote SMTP server connects to your external mail relays, that machine is supposed to wait until you issue the standard SMTP greeting (the thing we set with `SMTP_LOGIN_MSG`). Then you go through the `HELO`, `"mail from:"`, `"rcpt to:"`, etc as we described earlier.

Spammers, however, are usually interested in firing their email in as fast as possible. Many bulk-mailers will simply send the entire SMTP communication without waiting for acknowledgements from your mail relays (this is generally referred to as *slamming*). Since well-behaved MTAs should never do this, you can detect this behavior and use it as a mechanism for discarding spam.

Starting with Sendmail v8.13, you can set the `GreetPause` parameter in your `sendmail.cf` file to create a brief delay between your mail relay accepting the new SMTP connection and actually sending out the `SMTP_LOGIN_MSG`. If remote SMTP server starts sending you SMTP commands during this delay period, your mail relay will simply drop the SMTP connection and not accept the email.

In your macro configuration files, use the setting you see at the bottom of the slide. The value for the `greet_pause` macro is in milliseconds, so our setting of "1000" means to delay for one second. You don't want this delay period to be too long, because of course the delay adds latency to each new email message. Something between 1 and 5 seconds should be sufficient.

I recommend this setting for all relay servers that process incoming Internet email.



Access Database

- Lets you create a list of senders, domains, and/or IP addresses to block
- Also be used to "white list" addresses that would normally be blocked
- Declaration similar to `mailertable`, but also use "delay_checks" feature:

```
FEATURE(`access_db',  
`hash -o -T<TMPF> /etc/mail/access')  
FEATURE(`delay_checks', `friend')
```

The Access Database

Sendmail v8.9 also introduced the access database feature, originally just as a mechanism for sites to create a "blacklist" of senders that they didn't wish to receive email from. Senders could be identified by domain name, hostname, IP address, or a network block.

Over time, however, each new Sendmail release added additional functionality to the access database. Now you can use the access DB as a "white list" to allow certain email addresses you know you want to get email from to bypass your anti-spam checks. For example, my personal black lists are so draconian at this point, that I'm actually blocking entire spam-friendly countries from sending me email. On the other hand, I actually have friends in some of these countries and want to receive their emails. So I pop their email address into my access DB "white list" and all is well. You can even do this by *recipient* address, so your "abuse@sysiphus.com" email address can remain open and allow people to let you know when you're blocking their email in error.

The access database is just another one of those Unix DB style databases, like the `mailertable` feature we discussed earlier. As you can see on the slide, the declaration for the "access_db" feature is very similar to the declaration for the `mailertable` (including specifying a database type of "dbm" instead of "hash" if your system doesn't support Berkeley DB files). The "-T<TMPF>" option is required when declaring the access DB feature for Sendmail v8.12 and later, and means that Sendmail should defer any incoming email with a temporary failure code if the access database is inaccessible for some reason. This typically is not an issue when your access DB is in a Berkeley DB or DBM file on disk, but does become a factor when you start wanting to put your access

DB entries in an LDAP database or other external data store (and no, I'm not going to tell you how to do that in this class).

In addition to the "access_db" declaration, however, you also want to add the "delay_checks" feature. Normally, the access DB and other anti-spam checks would be consulted as soon as the remote server issues the "mail from:" command to set the message sender. The "delay_checks" feature tells Sendmail to wait until both the sender and recipient addresses have been sent before deciding whether or not the message is spam. This is how you make sure that your "abuse@sysiphus.com" address receives an unfiltered email stream while protecting all of your other users from spam.

Specifically, `FEATURE(`delay_checks', `friend')` means that the majority of your users want spam filtering but you need to declare certain addresses as being unfiltered. If you change the ``friend'` to ``hater'`, then you need to use the access DB to specifically indicate the users that you want to do spam filtering for, and then all other users receive unfiltered email feeds. The ``hater'` version is not a common posture for most organizations, but might be useful in an ISP environment where users "subscribe" to spam control as an optional feature.



Sample Access DB Entries

■ Very flexible feature set:

```
Spam:abuse@sysiphus.com    FRIEND
Connect:202.9.145.21      REJECT
Connect:63.99.230        REJECT
From:insidertrading.com    REJECT
From:zecpax@yahoo.com     ERROR: You annoy me
From:biz                  ERROR:550 Move out of biz
From:friend@some.biz      RELAY
GreetPause:192.168       0
```

■ Text file must be converted to Unix DB format with **makemap**:

```
cd /etc/mail
makemap hash access < access
```

Like the mailertable, you create your access DB entries in a text file—`/etc/mail/access` is the standard location, and the one we're using for our examples. There are lots of different things you can set in your access DB:

- The "Spam:... FRIEND" syntax you see in the first line of the example on the slide is how you mark certain email recipients like "abuse@sysiphus.com" as wanting to bypass your normal anti-spam filters. This is the functionality that is enabled by the "delay_checks" feature.
- Next you see how to filter incoming SMTP connections by IP address with the "Connect:" keyword. The first of these two entries blocks a specific IP address and the second blocks an entire network range. Again, Sendmail is unable to deal with networks that are non-octet-aligned, so for networks smaller than a /24, you end up having to list each IP address individually.
- The "From:" syntax allows you to block specific sender addresses, either by domain (insidertrading.com), email address (zecpax@yahoo.com), or even top-level domain—the last of the three examples blocks all addresses in the ".biz" domain. As you can also see in these examples, you are allowed to specify the error message you send back when you reject the email, including setting the SMTP return code.
- Of course, blocking everything under ".biz" might be something of a problem if a friend of yours has their email address in a ".biz" domain. You can use the "From:... RELAY" syntax to "white list" sender addresses that you want to bypass

your spam filters (similar to how "Spam:... FRIEND" bypasses spam filters for specific recipient addresses).

- If you enable both the GreetPause and access DB features, then you can also put "GreetPause:" tags in your access DB to override the default GreetPause value for specific hostnames, domains, IP addresses, and/or network ranges. This is often useful so you don't end up delaying email from your own internal users or from domains that you exchange a lot of email with. Note that to take advantage of this functionality you must declare the "greet_pause" feature *after* you declare "access_db". If you don't, you see the following warning when you run your macro file through m4:

```
$ m4 external.mc >external.cf
*** WARNING: FEATURE(`greet_pause') before
        FEATURE(`access_db')-- greet_pause will
        not use access_db!
```

Once you've got the /etc/mail/access text file set up, you use makemap to turn it into a Berkeley DB or DBM-style database file for Sendmail. And again, you can automate this process somewhat by creating a Makefile as we did for the mailertable. Here's a sample Makefile that will check both the access DB and your mailertable and rebuild either or both of them as necessary:

```
all:: access.db mailertable.db

access.db:: access
        makemap hash access < access

mailertable.db:: mailertable
        makemap hash mailertable < mailertable
```



DNS-Based Blacklists

- Spam fighting groups collect IP addresses of known spam emitters
- They then publish this information via special DNS zones
- "dnsbl" feature queries these zones to validate new SMTP connections
- Some lists are more "aggressive" than others (higher false-positive risk)
- Zones tend to move around due to "enemy action" by spammers

Automated DNS-Based Blacklists

The only problem with the access DB is that it relies on individual administrators to take time out of their busy lives to configure and maintain the entries. Wouldn't it be nice if somebody else took care of all that work for you?

There are many groups on the Internet today that hate spam so much that they spend a significant portion of their lives tracking down the servers that the spammers are using to send out their spam. Since these servers tend to move around, this is a real challenge. These spam-fighting groups publish the IP addresses of these known spam servers in special DNS maps.

The Sendmail "dnsbl" (DNS BlackList) feature allows you to query these special DNS zones in real time for every new SMTP connection received by your mail servers. If the IP address of the remote end of the connection appears in one of the dnsbl maps that you subscribe to, then Sendmail will simply drop the incoming SMTP connection and not accept the email.

Of course the issue here is that you're now relying on some external entity to manage your blacklist. What are their criteria for blacklisting various IP addresses? How big a risk of false-positives do you have? Certain blacklists will carefully verify that the sender really is responsible for a given piece of spam or is an open relay before adding that IP address to their blacklist. Other blacklist maintainers have a "shoot first and ask questions later" attitude. You should carefully read the information at the blacklist maintainers' web sites before subscribing (see next slide for a list).

The other problem is that the spammers *hate* these lists. Many spammers will try and shut down these lists with threats of legal action and/or denial-of-service attacks. So you have to keep an eye on the various lists that you subscribe to and make sure that they continue to function. Often when the spammers manage to shut down one list it reappears under a different name someplace else (usually outside the US where it's harder to bring legal action).

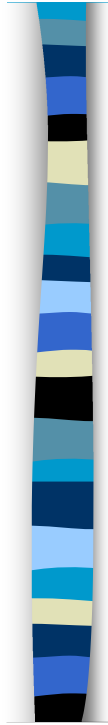


Popular dnsbl Sites

```
FEATURE(dnsbl, `bl.spamcop.net',  
        `Rejected- see http://spamcop.net/')  
FEATURE(dnsbl, `sbl-xbl.spamhaus.org',  
        `Rejected- see http://www.spamhaus.org')  
FEATURE(dnsbl, `dnsbl.njabl.org',  
        `Rejected- see http://njabl.org/')  
FEATURE(dnsbl, `list.dsbl.org',  
        `Rejected- see http://dsbl.org/')  
FEATURE(dnsbl, `relays.ordb.org',  
        `Rejected- see http://www.ordb.org/')
```

The slide shows five of the more popular dnsbls and how to configure Sendmail to subscribe to each blacklist. I personally use all five of these dnsbls on my external email relay for Deer Run. Yes, that means up to five additional DNS lookups for each incoming SMTP connection, which consume network bandwidth, processing resources, and time, but it's worth it to me because these lists really block a lot of spam. These lists also generate a lot of false-positives, which is why I have extensive "white lists".

Notice that the dnsbl declarations allow you to specify an error message that you want Sendmail to send to the remote site when you drop their SMTP connection. This error message is supposed to contain a pointer to the web site of the dnsbl maintainer, so that the sender of the message can learn how to get themselves de-listed if they believe that they've been listed in error.



Milters Extend Sendmail

- Milter is an API that defines how Sendmail can talk to external programs
- Milters can do anything, but are commonly used for spam/virus control
- Must ensure your Sendmail binary is compiled with Milter support
- Then add Milter hooks in `.mc` file:

```
INPUT_MAIL_FILTER(`greylist',  
  `S=local:/var/greylist/greylist.sock, F=T')  
INPUT_MAIL_FILTER(`mimedefang',  
  `S=unix:/var/Defang/defang.sock, F=T, T=S:1m;R:1m')  
define(`confMILTER_MACROS_CONNECT', `j, {if_addr}')  
define(`confMILTER_MACROS_ENVFROM', `i')
```

The Sendmail Milter Interface

Milter Basics

Sendmail is already an extremely complicated program, and all of the ideas that people were coming up with to fight spam were making Sendmail even more complicated. Plus all of these new spam fighting techniques were creating a whole bunch of conflicting updates and modifications that were not officially "supported" by the Sendmail maintainers. So the idea was to create an API that developers could use to create external programs that Sendmail could talk to in a clearly defined way. This was the genesis of Sendmail's "Milter" (Mail Filter) interface. Milter support was available as early as Sendmail v8.10, but it wasn't until Sendmail v8.12 that it became tightly integrated into the Sendmail distribution.

To begin using the Milter interface, you need to make sure that your Sendmail binary was compiled with Milter support. Remember how we used the "`-d0.1`" option to check the version number of our Sendmail binary? Well the same command can tell you whether your binary was compiled with Milter support:

```

$ /usr/sbin/sendmail -d0.1 </dev/null
Version 8.13.4
  Compiled with: DNSMAP HESIOD HES_GETMAILHOST LDAPMAP LOG
                MAP_REGEX MATCHGECOS MILTER MIME7TO8 MIME8TO7
                NAMED_BIND NETINET NETINET6 NETUNIX NEWDB NIS
                PIPELINING SASLv2 SCANF SOCKETMAP STARTTLS
                TCPWRAPPERS USERDB USE_LDAP_INIT

```

```

===== SYSTEM IDENTITY (after readcf) =====
      (short domain name) $w = internal
      (canonical domain name) $j = internal.sysiphus.com
      (subdomain name) $m = sysiphus.com
      (node name) $k = internal.sysiphus.com
=====

```

Recipient names must be specified

See the word "MILTER" that's highlighted on the third line of output? That's your indication that the binary was compiled with Milter support.

If you don't see that keyword, then you need to rebuild your binary—or download the source distribution and build your own binary if you've been using the Sendmail binary provided by your vendor. More information on building Sendmail with Milter support can be found in the `libmilter/README` file in the Sendmail source distribution, but basically the idea is that you create a file called `devtools/Site/site.config.m4` in the Sendmail source distribution that contains this line:

```
APPENDDEF(`conf_sendmail_ENVDEF', `-DMILTER')
```

The run the `Build` script as normal to build the Sendmail binary.

Once you have Milter support compiled into Sendmail, you will need to hack your macro configuration files to tell Sendmail how to talk to the various Milters you've installed. The example on the slide is just to give you a taste of what these declarations typically look like. I'm not going to go into much detail on what the syntax means because usually you just cut-and-paste the Milter declarations from the installation instructions for the particular Milter that you're installing.

If you're interested in writing your own Milters, the `libmilter/README` file in the Sendmail source distribution has some pointers and sample code, and there's plenty more information at www.milter.org.



MIMEDefang Milter

- MIMEDefang is a "framework" where you can snap in different functionality
- SpamAssassin is a popular tool for recognizing and tagging possible spam
- ClamAV is popular free antivirus filter, but commercial tools also supported
- Pulling together and maintaining these Open Source utilities is non-trivial

The MIMEDefang Milter

One of the more popular Open Source Milters is a tool called MIMEDefang (www.mimedefang.org), written and maintained by the folks at Roaring Penguin software (who also incorporate the same technology into their CanIt™ line of anti-spam products). In a sense, MIMEDefang doesn't really "do anything" by itself. It's designed as a framework that you can snap other tools into in order to handle spam and virus filtering.

The most common Open Source anti-spam solution that people use with MIMEDefang is Spamassassin (spamassassin.apache.org). The Spamassassin developers have created a fairly complicated set of heuristics for recognizing spam based on a wide variety of different technologies (blacklists, statistical filters, etc) that boil down to a "score" that indicates how likely the message is to be spam. Normally the message is tagged with this score and then the site can decide to simply reject messages that score higher than a certain value, or allow their users to individually filter messages above a certain score. If Spamassassin gives a message a score of 5 or more, it's probably spam.

On the anti-virus front, the most popular Open Source tool is called ClamAV (www.clamav.net), although MIMEDefang also has built-in hooks for many of the popular commercial anti-virus tools (like Trend Micro, Symantec, etc). The nice thing about ClamAV is that their "virus" database also contains signatures for many common Phishing scams so that you can recognize and eliminate this email before it gets to your users. There have been a lot of security updates to ClamAV recently (frankly, it's been kind of hard to keep up with all of the updates), but what that tells me is that the software is effective enough that the bad guys are really spending a lot of time trying to figure out how to subvert ClamAV. Once a virus has been recognized, MIMEDefang allows you to quarantine viruses, etc but the easiest thing to do is often just to have MIMEDefang

remove the attachment that contains the virus and replace it with a message explaining what happened.

There's a decent How-To document on installing MIMEDefang with ClamAV and Spamassassin at <http://www.mickeyhill.com/mimedefang-howto/>. I will warn you, however, that MIMEDefang, ClamAV, and Spamassassin all depend on lots of different Perl modules and Open Source software packages. Getting all of this software together (particularly if you're installing it on a non-Linux platform) and then configuring all of the various components is a huge pain in the rump.



Content Signature Databases

- Organizations maintain spam traps to create databases of known spam
- Client mail servers update local databases regularly ala virus definitions
- Query databases via Milter, discarding future appearance of similar messages
- Can find both free (Vipul's Razor) and commercial (Brightmail) solutions

At least it prevents you from getting spammed with the same thing twice...

Content Signature Databases

You can think of content signature databases as sort of an anti-virus-like approach to spam. Basically, the organizations creating these databases create dummy email accounts as "spam traps" to try and collect new forms of spam. The spam is analyzed and a signature is created that recognizes future copies of that spam as well as other similar sorts of spam messages. These signatures are collected into a database that you can download via a regular update mechanism to your mail servers. You run a Milter program on your mail server that analyzes incoming email and compares it to the signatures in the database—if there's a match, either drop the message or tag it as spam by adding a special header.

There's an Open Source implementation of this idea called Vipul's Razor (razor.sourceforge.net). The popular commercial implementation was developed by Brightmail, now owned by Symantec (<http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=642>).



Greylisting

- Normal mail servers will try to resend messages after temporary failures
- Spam is commonly sent from systems that never try to resend failed email
- Greylisting tells server to defer the first email message from any new sender
- When the remote server resends the message, that server gets "white listed"
- Spam is not resent, so it never gets in!

Greylisting

When a normal mail server is unable to deliver an email message for some reason it will normally queue that message and attempt delivery at a later date. Spammers, however, will normally use bulk-emailing tools that never re-attempt to deliver email—in fact these tools rarely ever track that the email was rejected in the first place. Greylisting (<http://projects.puremagic.com/greylisting/>) is an anti-spam technique that (like GreetPause) uses incorrect behavior on the part of spammers to stop spam mail while allowing proper MTAs to function.

The idea is that greylisting tracks "triples" of the sender address, recipient address, and IP address of the remote SMTP server for each incoming email message. If the greylisting Milter has not yet seen an email that matches the "triple" for a given incoming message then the greylisting Milter instructs Sendmail to send the remote SMTP server a 4xx "defer" message. This should cause the remote SMTP server to queue the message for later delivery. When the delivery is attempted again, the greylisting Milter recognizes that it's seen the "triple" before and allows the email to pass (after which the message is subjected to the rest of your anti-spam filters). Typically the greylisting Milter will also automatically "white list" the remote SMTP server for some period of time so that you don't delay any further emails from that IP address—because spammer or not the remote mail server is obviously well-behaved and there's no point in delaying further emails.

Greylisting is a great idea because it precisely targets the spammers without much impact on normal email traffic, and it's essentially self-maintaining. Frankly, it's probably the most useful anti-spam technique to be developed in the last few years. Personally, I use the `milter-greylst` implementation (<http://hcpnet.free.fr/milter-greylst/>) on my mail servers, although there are several other greylisting implementations available.



Known Issues With Greylisting

- Delayed delivery on first message
 - *Try to get over it...?*
- "Farms" of outgoing mail servers
 - *Simple heuristics cover most cases*
- Broken outgoing mail and list servers
 - *White list the ones you have to talk to*
- Spammers are going to get smarter
 - *Yep, it's an arms race...*

However, there are some known issues with greylisting that need to be dealt with:

- Perhaps the most obvious issue is that the first email messages from a given server will get delayed by your greylisting system until one of the messages is resent and the server gets white listed. You want to increase the time servers remain on your automatic "white list" (I use 30 days) and reduce the default deferral time for messages. The standard greylist implementation is to defer messages for one hour, but frankly five minutes is enough to make sure you're talking to a legitimate mail server. Still it may take an hour or more for the remote server to retry delivery ("-q1h", remember?) so that first email can take a long time. Ultimately your users have to recognize that however much they would like it to be so, email is not an instantaneous communications mechanism. Let them IM.
- The other problem you run into is large ISPs that have "farms" of outgoing mail servers. The machine that tried to send you the email initially may not be the same machine that does the retry. But as far as your greylisting implementation is concerned, you have two different "triples" because the remote IP address is different on each attempt. The easiest thing to do here is to not track the exact IP address of the sending machine, but instead use just the first three octets. In most cases all of the machines in the outgoing message farm at the remote ISP are in the same narrow range of IP addresses.
- Certain broken mailing list servers actually put a different sender address on every outgoing email and retry attempt (they do this to try and track bounce messages that they get back when trying to deliver a given email message). Unfortunately this hoses greylisting because you're dealing with a different "triple" on every retry. The

only thing you can do here is track these list servers and manually put them in a permanent "white list" so that they're ignored by your greylisting software. The greylisting software I've seen always comes pre-configured with a "white list" of popular list servers that exhibit this behavior.

Ultimately, of course, the spammers are going to wise up and resend their spam in order to get around your greylisting. Of course, if they do this then you can track the IP address of the spammer's mail server and block it—in fact your `dnsbls` might get updated with the spammer's IP before the spammer gets the opportunity to resend their spam. Also, having to resend their messages increases the cost to the spammer for sending out their spam. Remember the goal is to make spamming economically less attractive to the spammers.



Too Many Choices!

- Outsourcing is a viable option
- "Safe" starting point for DIY types:
 - Enable "GreetPause" feature
 - Use Greylisting
 - MIMEDefang with Spamassassin & ClamAV
 - Enable access DB as a backup
- Either way, you'll need to teach your users how to filter tagged spam

Now I'm Really Confused

The good news is that you have a wide array of options for spam and virus control. The bad news is that choosing the appropriate solution for your organization is becoming increasingly complicated—to say nothing of the complexity of implementing your chosen solution. If you don't want to spend the resources on becoming a spam-fighting expert, you can always outsource the job to somebody else.

If you're willing to devote the time and resources, you can build an effective, "almost free" (the only thing you pay for is the hardware and your time) anti-spam and anti-virus solution using Linux and Open Source tools. Most sites will find a combination of GreetPause, greylisting, and MIMEDefang+Spamassassin+ClamAV will do an excellent job eliminating or at least properly tagging 95-99% of your incoming spam volume. False-positives should be almost non-existent. You can always use the access DB to manually filter persistent spammers who don't get caught by your automatic filters. The next step up from there is `dnsbls`, but that also greatly increases your false-positive risk.

Whether you go the outsourcing route or use Spamassassin, you will have to configure your users' mail clients to filter the tagged email into folders. And you'll have to teach your users how to go into those folders and retrieve any legitimate email that was mis-tagged as spam. Teaching them how to modify their filters to avoid mis-tagging this email in the future (or at least how to report the emails to you so that you can manage this process) is also a good idea.

Exercises

6.1. Simple Spam Control

Exercise Goal – Get some experience with the GreetPause and access DB features. Learn different mechanisms for testing your anti-spam filters.

6.1.1. Modify the configuration on your external server to set a GreetPause value of 1.5 seconds and also activate the access DB and "delay_checks" features. Create an access DB that prevents email to the abuse@sysiphus.com address from being filtered but which blocks email from senders in the domain foo.biz to all other recipient addresses. Restart your Sendmail daemon to pick up the changes. Take notes on your configuration settings in the space below.

Hint: Don't forget to declare "greet_pause" after "access_db"!

6.1.2. Now we're going to test if our `GreetPause` setting is functioning with a little help from the "nc" (netcat) command. In your home directory you should find a file called "smtp-session" that contains all of the SMTP commands necessary to transmit a simple email message. Pretend you're a "slammer" and use the following nc command line to fire the commands at your external relay:

```
cat smtp-session | nc ext1.sysiphus.com 25
```

The actual hostname you use in the above command may vary depending on which server you're using for your tests. Compare the output of the above command with the same command run against one of the internal relay servers that doesn't have `GreetPause` enabled. Write down your observations in the space below.

6.1.3. Now let's test our access DB entries. Forge a piece of email claiming to be from some sender in the `foo.biz` domain and use the recipient address of `hal@sysiphus.com`. Did the "right thing" happen in this case? Now try the same experiment except use the recipient address of `abuse@sysiphus.com`. Now what happens and is it the expected behavior? Use the space below to take notes on what you observe.

Answers to Exercises

6.1. Simple Spam Control

6.1.1. Modify the configuration on your external server to set a `GreetPause` value of 1.5 seconds and also activate the access DB and "delay_checks" features. Create an access DB that prevents email to the `abuse@sysiphus.com` address from being filtered but which blocks email from senders in the domain `foo.biz` to all other recipient addresses. Restart your Sendmail daemon to pick up the changes. Take notes on your configuration settings in the space below.

Here are the relevant macros for your `external.mc` file:

```
FEATURE(`access_db',
        `hash -o -T<TMPF> /etc/mail/access')
FEATURE(`delay_checks', `friend')
FEATURE(`greet_pause', `1500')
```

Remember that you want to declare "greet_pause" after "access_db" so that you can take advantage of the "GreetPause:" hook in your access DB.

According to the instructions in the exercise, your `/etc/mail/access` file should look like this (although the order of the entries is not important):

```
Spam:abuse@sysiphus.com  FRIEND
From:foo.biz             REJECT
```

The rest of the configuration process should go something like this:

```
$ m4 external.mc > external.cf
$ /bin/su
Password: <not echoed>
# cp external.cf /etc/mail/sendmail.cf
# cd /etc/mail
# makemap hash access < access
# /etc/init.d/sendmail restart
Shutting down sendmail:           [ OK ]
Shutting down sm-client:         [ OK ]
Starting sendmail:                [ OK ]
Starting sm-client:              [ OK ]
```

6.1.2. Now we're going to test if our `GreetPause` setting is functioning with a little help from the "nc" (netcat) command. In your home directory you should find a file called "smtp-session" that contains all of the SMTP commands necessary to transmit a simple email message. Pretend you're a "slammer" and use the following nc command line to fire the commands at your external relay:

```
cat smtp-session | nc ext1.sysiphus.com 25
```

The actual hostname you use in the above command may vary depending on which server you're using for your tests. Compare the output of the above command with the same command run against one of the internal relay servers that doesn't have `GreetPause` enabled. Write down your observations in the space below.

Here's the test against the external server with `GreetPause` enabled:

```
$ cat smtp-session | nc ext1.sysiphus.com 25
554 ext1.sysiphus.com ESMTP not accepting messages
250 ext1.sysiphus.com Hello [192.168.1.1], pleased to...
550 5.0.0 Command rejected
550 5.0.0 Command rejected
550 5.0.0 Command rejected
500 5.5.1 Command unrecognized: "From: hal@deer-run..."
500 5.5.1 Command unrecognized: "To: hal@sysiphus.com"
500 5.5.1 Command unrecognized: "Subject: testing"
500 5.5.1 Command unrecognized: ""
500 5.5.1 Command unrecognized: "1 2 3"
500 5.5.1 Command unrecognized: "."
221 2.0.0 ext1.sysiphus.com closing connection
```

Notice the "not accepting messages" warning and all of the "Command rejected" and "Command unrecognized errors".

Now let's try that against one of the internal servers:

```
$ cat smtp-session | nc int1.sysiphus.com 25
220 int1.sysiphus.com ESMTP Sendmail 8.13.4/8.13.4...
250 int1.sysiphus.com Hello [192.168.1.1], pleased to...
250 2.1.0 hal@deer-run.com... Sender ok
250 2.1.5 hal@sysiphus.com... Recipient ok
354 Enter mail, end with "." on a line by itself
250 2.0.0 k059BPhT004045 Message accepted for delivery
221 2.0.0 int1.sysiphus.com closing connection
```

That seems to have gone a lot better!

6.1.3. Now let's test our access DB entries. Forge a piece of email claiming to be from some sender in the `foo.biz` domain and use the recipient address of `hal@sysiphus.com`. Did the "right thing" happen in this case? Now try the same experiment except use the recipient address of `abuse@sysiphus.com`. Now what happens and is it the expected behavior? Use the space below to take notes on what you observe.

Normally we would expect our access DB to reject email where the sender is in the `foo.biz` domain:

```
# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
220 mailer ESMTP ready
helo localhost.localdomain
250 ext1.sysiphus.com Hello localhost.localdomain ...
mail from: spam@foo.biz
250 2.1.0 spam@foo.biz... Sender ok
rcpt to: hal@sysiphus.com
550 5.7.1 hal@sysiphus.com... Access denied
quit
```

Notice, however, that our message wasn't rejected until we actually entered the recipient address. This is the effect of the "delay_checks" feature.

Now let's try using the same sender address but changing the recipient to `abuse@sysiphus.com`. Our "Spam:...FRIEND" setting in the access DB should allow this email to pass:

```
# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
220 mailer ESMTP ready
helo localhost.localdomain
250 ext1.sysiphus.com Hello localhost.localdomain...
mail from: spam@foo.biz
250 2.1.0 spam@foo.biz... Sender ok
rcpt to: abuse@sysiphus.com
250 2.1.5 abuse@sysiphus.com... Recipient ok
quit
```

Yep, that worked. Notice that I didn't bother to complete the message transmission—I just used the "quit" command as soon as I verified that the access DB wasn't blocking my transmission.

Virtual Domains



Virtual Domains

This module deals with Sendmail functionality for managing multiple email domains on a single mail server. After completing this module you should:

- Understand the distinction between "parallel" and true virtual domains, and know the appropriate configuration to use in each case
- Have some pointers on how to configure DNS for your virtual domains
- Know how to enable and configure Sendmail's `domaintable` and `virtusertable` features



What are We Talking About?

- It's not unusual for sites to have to deal with multiple domain names
- Managing dedicated mail servers for each domain doesn't scale
- Sendmail provides a couple of different mechanisms for dealing with this
- Which one you choose depends on what sort of email behavior you need...

What Are We Talking About?

These days it's not uncommon for a given organization to handle email for many different domains. Your marketing department may have run amok and registered every possible misspelling of your domain name in .com, .net, .org, and .biz and then told you that you have to be willing to accept email for any and all of those domains. Or your company may have acquired another company and you have to route email for the acquired domain. Or you're an ISP/hosting provider who provides web and email hosting. Or...well, let's just say there are lots of reasons.

If you were insane you could set up a separate email server for each domain, but that's obviously not a scalable solution. What you want is to be able to properly handle email for an arbitrary number of domains on a single mail server. Luckily, Sendmail provides a couple of different features for doing just that. Which feature you choose depends primarily on the email behavior you want for a given domain name. We'll discuss these issues in more detail on the next couple of slides.



"Parallel" Domains

- When then new domain should behave just like your primary email domain
- Typical circumstances:
 - Acquiring another company
 - Marketing department goes wild
 - "Misspelling" type domains
- Use Sendmail's "`domaintable`" feature to map these domains to your primary
- All addresses will be rewritten to primary domain

"Parallel" Domains

"Parallel domains" is a term I coined for the situation where you have multiple email domains that are all essentially interchangeable. The behavior for the email address "user@domain1.com" should be the same as the behavior for "user@domain2.com", "user@domain3.com", and so on.

Most often, this situation is a gift from your marketing department when they decide to register all of those different permutations of your regular domain name in order to avoid people "squatting" on similar domain names, or to help users who are unable to correctly spell your regular domain name. You also often end up in this situation 6-12 months after acquiring another company. By that point, the new company has been integrated enough that their old email infrastructure has been absorbed into the parent company—you don't want to bounce email to the old domain, but you no longer need special case routing rules for it either.

This is what Sendmail's "`domaintable`" feature was actually designed for. Remember that the `domaintable` maps addresses from one domain to another. What I didn't mention when we talked about this feature earlier is that the `domaintable` actually rewrites all of the addresses that match in your `domaintable` to the canonical domain—both "From" and "To" and in both the message itself and the envelope. This is normally what you want in this case.

We discussed the declaration of the `domaintable` feature and configuration of the `domaintable` database earlier in the course, so I won't belabor the issue. What's worth noting about the `domaintable` feature is that if you're using it on your external relays

you don't need to list your "parallel" domain names in either the `local-host-names` file or the `relay-domains` file. Sendmail recognizes the domains listed in the `domaintable` file as being local. Of course, if you're using the `domaintable` feature on your internal mail server then you'll have to modify your external mail relay configuration to pass email for these domains. This would mean adding rules to your `mailertable` and updating your `relay-domains` file.



True Virtual Domains

- When the new domain needs to have different routing rules than the primary
- Typical circumstances:
 - ISPs and hosting providers
 - Special marketing promotions
 - Outsourcing deals
- Use "**virtusertable**" feature to create database of special rules
- Just does routing-- will not modify any addresses (unlike **domaintable**)

Virtual Domains

A true virtual domain is one that has completely different mail routing behavior from your normal canonical email domain. ISPs and hosting providers will often support virtual email domains for their customers. Personally, I host a number of email domains for "friends and family". I've also used virtual domain functionality to route email for special promotions, like when your marketing people want you to do something intelligent with the email for the "ourproductkicksseriousbutt.com" domain they've registered. You might even use the virtual domain feature to handle email routing for portions of your own domain—like when you outsource `support.yourdomain.com` to another company. You can also use `mailertables` for this in most cases, however.

Virtual domains are handled by Sendmail's `virtusertable` feature. Yes, it's just another Unix DB style database. With the `virtusertable` you can associate either individual addresses in the virtual domain or all addresses in the domain (or even a combination of the two) with email addresses in another domain. For example you might tie `sue@virtual.com` to Sue's actual email address `BoyNamedSue@AOL.com`. Unlike the `domaintable` feature, when you use `virtusertable` Sendmail does not do any address rewriting in the body of the message itself—it leaves all of the original mail headers alone. The only thing that gets changed is the recipient address on the message envelope so that the original message can be routed to its new destination.



Where to Deploy

- It's possible to use this functionality on either your internal or external relay
- Personally, I find it more convenient to manage them on the external relay
- You must make sure you have DNS set up properly for each domain...

Deployment Issues

Where To Deploy

Frankly, you can deploy these virtual domain features on either your internal or external relays. Personally, it makes more sense to me to deploy them on your external relay servers:

- In the case of the `domaintable` feature, having the `domaintable` on your external relay means all of the addresses in the message get rewritten to your canonical domain *before* they get into your internal email infrastructure. That means you only have to configure your internal mail servers to recognize your single canonical email domain, or it least it significantly reduces the number of email domains your internal mail servers have to deal with.
- It's likely that your `virtusertable` will route at least some of the email for the virtual domains to users at other organizations or ISPs. There's no point in relaying the email all the way in to your internal email infrastructure if you're just going to have to send it right back out again.

As always, your mileage may vary. Deploy these features wherever it makes the most sense to you.



DNS Configuration Notes

- Basically you need to make sure there's a proper MX record in each domain:

```
virtual.dom. IN MX 10 external.sysiphus.com.
```

- If all zones share the same web and mail servers, can use a single zone file:

```
$TTL 24h
@ IN SOA ns.sysiphus.com. hostmaster.sysiphus.com. (
    2005122000 24h 2h 7d 2h )
@ IN NS ns.sysiphus.com.
@ IN NS ns1.granitecanyon.com.
@ IN MX 10 external.sysiphus.com.
@ IN A 209.218.104.48
www IN A 209.218.104.48
```

DNS Configuration

You will need properly configured DNS for each of the virtual domains you own—at least as far as having an MX record that routes email for the domain to your external mail relays. You'll also probably want to set up a record so that people can find the web server for the domain.

In the case where you're using the same web server for all of the different virtual domains (easy to do with Apache), DNS configuration is straightforward. In fact, you can use the same DNS zone file for all of your virtual domains—at least if you're using BIND on Unix for your DNS server. Take a look at the sample zone file on the slide. The trick is that the "@" character is a special macro in DNS zone files that gets expanded by BIND with the domain name for the zone from BIND's `named.conf` file. So if you had this zone file installed as "virtual-domains" in your default name server directory, you could write your `named.conf` file as follows:

```
zone "virtual1.com" {
    type master;
    file "virtual-domains";
};

zone "virtual2.com" {
    type master;
    file "virtual-domains";
};
```

Notice we're referencing the same "virtual-domains" file in each declaration. Why does this work? When named reads the declaration for the "virtual1.com" zone, it reads the virtual-domains file and expands all of the "@" markers with "virtual1.com". When named reads the declaration for "virtual2.com" it re-reads the "virtual-domains" file and replaces all of the "@" signs with "virtual2.com". Slick, huh? Since the "www" name in the zone file doesn't have a trailing "." on it, named automatically appends the correct domain name to this name as well, making it "www.virtual1.com" the first time through the file, and "www.virtual2.com" the second time it reads the file.

Note that it's perfectly fine that the NS and MX records for each zone point to servers in a different domain. In fact, this is common behavior for many domains registered on the Internet today.



virtusertable Feature

- Add two directives to your `.mc` file:

```
VIRTUSER_DOMAIN_FILE(`-o /etc/mail/vdoms')  
FEATURE(`virtusertable',  
        `hash -o /etc/mail/virtusertable')
```

- The `vdoms` file is a text file only (like `local-host-names`)
- The `virtusertable` file needs to be converted using `makemap`

The virtusertable Feature

Setting up the `virtusertable` feature is only slightly more complicated than setting up the `domaintable`. As you can see in the example on the slide, the `virtusertable` declaration itself is just like the `domaintable` and `mailertable` declarations we've seen already.

However, there's also the `VIRTUSER_DOMAIN_FILE` setting that defines a file that acts a lot like `local-host-names`, except that the `VIRTUSER_DOMAIN_FILE` lists the virtual domains you want this machine to handle. For purposes of promiscuous relay checks, the domain names in the `VIRTUSER_DOMAIN_FILE` will be treated as being "local", just like those in `local-host-names`. But in terms of mail routing, addresses in the domains listed in the `VIRTUSER_DOMAIN_FILE` will be handled by your `virtusertable` database.

You can call your `VIRTUSER_DOMAIN_FILE` anything you want, although `/etc/mail/virtuserdomains` is traditional. Frankly, this file name didn't fit on the slide real well, so I shortened the file name. Notice also that you're allowed to include the `"-o"` option even for these simple text files (just like we're doing for the `virtusertable` DB file) so that Sendmail will start normally even if the `vdoms` file doesn't exist.

The thing to remember here is that your `VIRTUSER_DOMAIN_FILE` is just simple text file with a list of your virtual domains, just like `local-host-names`. The `virtusertable` file is the text file that you edit to do the actual virtual domain mappings and then convert into a Unix DB file using `makemap`.



virtusertable DB Entries

- Can map individual addresses, entire domains, or a combination of the two:

```
@fictional.com      bob@sysiphus.com
abuse@virtu.com    postmaster@sysiphus.com
postmaster@virtu.com postmaster@sysiphus.com
@virtu.com         alice@sysiphus.com
```

- Nested mappings are allowed (though I'm not sure why you'd want this):

```
support@virtu.com  support@responder.com
@responder.com     resp@sysiphus.com
```

As you can see in the first example on the slide, the format of the `virtusertable` database itself is a bit more complicated than the `domaintable`. In the first line of the example, we're mapping all email addresses under `fictional.com` to `bob@sysiphus.com`. In the next two lines we're mapping specific email addresses—the `abuse@` and `postmaster@` addresses for `virtu.com`—to the `postmaster@sysiphus.com` address. The last line takes all other addresses in the `virtu.com` domain and routes them to `alice@sysiphus.com`. The point is you are allowed to mix routing rules for specific email addresses in a given domain with a "catch all" rule for handling other email addresses that you didn't specify.

Sendmail will do multiple lookups in the `virtusertable` as necessary. So you could do something like the second example on the slide where you have one rule that routes email to an address in another virtual domain configured in the same DB file. I find this kind of configuration confusing, however. My recommendation is to always use the real "canonical" email address of the recipient on the righthand side of the `virtusertable` database, just to avoid confusion.

Once you've got the `virtusertable` set up the way you want, it's time to run `makemap`. If your external mail relays have a `mailertable`, an `access DB`, a `domaintable`, and a `virtusertable` as my mail server does, then that trick I showed you of creating a `Makefile` to automate rebuilding these databases is really, really helpful. Here's a sample `Makefile` that will handle rebuilding all of these databases as necessary:

```
all:: mailertable.db access.db \  
    domaintable.db virtusertable.db  
  
access.db:: access  
    makemap hash access < access  
  
domaintable.db:: domaintable  
    makemap hash domaintable < domaintable  
  
mailertable.db:: mailertable  
    makemap hash mailertable < mailertable  
  
virtusertable.db:: virtusertable  
    makemap hash virtusertable < virtusertable
```

If you install this file as `/etc/mail/Makefile`, then whenever you type `make` in the `/etc/mail` directory the appropriate databases will automatically get rebuilt if the text file was modified more recently than the `.db` file used by Sendmail.



One Last Thought...

- Sometimes you acquire a company that keeps its own identity, mail servers, etc
- However, you don't want email to route to this company via the Internet
- You can still use `mailertable` on your internal relay to route email as needed
- Maybe via a WAN or VPN link to other firm's internal mail servers...

Other Foreign Domains

Sometimes you're in a situation where your organization has multiple divisions, each with its own permanent Internet domain identity, mail servers, etc. You can also end up in this situation immediately after you've acquired another company, when they're still maintaining their own independent email infrastructure. In these cases, there's often a lot of information of a sensitive and proprietary nature flying back and forth between these organizations—information that you don't necessarily want to traverse the Internet for delivery.

Always remember that you do have the `mailertable` feature for your internal relay servers. So rather than treating the other division's domain as just another domain that needs to get forwarded to the external relays for Internet delivery, you could always put in a special rule into your `mailertable` like:

```
other.com      smtp:internal.other.com
```

At this point, it's up to your internal network routing to make sure that the packets to the machine `internal.other.com` do not traverse the Internet. Maybe you've got a point-to-point WAN link set up between the two organizations, or maybe you have a dedicated VPN connection between the two sites.

Exercises

7.1. Virtual Domains

Exercise Goal – Gain some experience with the `virtusertable` and `domaintable` features, and examine the different behaviors of each.

By the end of this exercise you'll have a `mailertable`, an access DB, a `domaintable`, and a `virtusertable` on your external relay server—I've included a `Makefile` in your home directory that will make it easier for you to keep the DB files up-to-date.

7.1.1. Add the `domaintable` and `virtusertable` declarations to your external relay config, install the new `.cf` file, and restart your MTA daemon. Write down the macro declarations you use in the space below.

Hint: Don't forget `VIRTUSER_DOMAIN_FILE`!

7.1.2. Create a `virtusertable` entry that maps the "virtual.com" domain to the address `marketing@sysiphus.com`. Create a `domaintable` entry that maps "parallel.com" to `sysiphus.com`. Use the `Makefile` provided in your home directory to create the DB files. Take notes on your configuration in the space below.

Hint: See previous hint...

7.1.3. Send a piece of email through your external mail gateway to a user in the `virtual.com` domain and another piece of email to a user in the `parallel.com` domain. Write down the queue IDs for each message in the space below—we're doing to want to look at the log entries associated with these messages in the next exercise.

7.1.4. Use the queue IDs from the previous exercise to pull out the log entries related to the messages you sent. What do you notice about the recipient addresses in the log entries? Make notes in the space below:

7.1.5. Now take a look at the "virtual-samples" file in your home directory, which contains copies of sample emails sent to the `parallel.com` and `virtual.com` domains. Compare the headers of each email—particularly the "Received:" and "To:" headers. What do you notice about each email? Make notes in the space below.

Answers to Exercises

7.1. Virtual Domains

7.1.1. Add the `domaintable` and `virtusertable` declarations to your external relay config, install the new `.cf` file, and restart your MTA daemon. Write down the macro declarations you use in the space below.

Here are the macros:

```
VIRTUSER_DOMAIN_FILE(`-o /etc/mail/virtuserdomains')
FEATURE(`virtusertable',
        `hash -o /etc/mail/virtusertable')
FEATURE(`domaintable',
        `hash -o /etc/mail/domaintable')
```

Notice that I'm using the full `/etc/mail/virtuserdomains` file name rather than the abbreviated "vdoms" name I used in the course slides. The longer name is the "standard" for this file.

And of course we have the standard install and update process:

```
$ m4 external.mc > external.cf
$ /bin/su
Password: <not echoed>
# cp external.cf /etc/mail/sendmail.cf
# /etc/init.d/sendmail restart
Shutting down sendmail:           [ OK ]
Shutting down sm-client:         [ OK ]
Starting sendmail:                [ OK ]
Starting sm-client:               [ OK ]
```

/Answers to Exercises continue on next page...]

7.1.2. Create a virtusertable entry that maps the "virtual.com" domain to the address marketing@sysiphus.com. Create a domaintable entry that maps "parallel.com" to sysiphus.com. Use the Makefile provided in your home directory to create the DB files. Take notes on your configuration in the space below.

The virtusertable entry is straightforward:

```
@virtual.com          marketing@sysiphus.com
```

However, you also need to remember to put the domain "virtual.com" in your /etc/mail/virtuserdomains file.

The domaintable entry is also trivial:

```
parallel.com          sysiphus.com
```

Once you have the text files set up, you should be able to cd to your home directory and run the following commands to copy and use the Makefile:

```
# cp Makefile /etc/mail
# cd /etc/mail
# make
makemap hash domaintable < domaintable
makemap hash virtusertable < virtusertable
```

/Answers to Exercises continue on next page...]

7.1.3. Send a piece of email through your external mail gateway to a user in the `virtual.com` domain and another piece of email to a user in the `parallel.com` domain. Write down the queue IDs for each message in the space below—we're doing to want to look at the log entries associated with these messages in the next exercise.

Here's the output of the SMTP session I used to generate my test cases:

```
$ telnet localhost 25
[...some output not shown...]
220 mailer ESMTP ready
helo localhost.localdomain
250 ext1.sysiphus.com Hello localhost.localdomain ...
mail from: sender@deer-run.com
250 2.1.0 sender@deer-run.com... Sender ok
rcpt to: hal@virtual.com
250 2.1.5 hal@virtual.com... Recipient ok
data
354 Enter mail, end with "." on a line by itself
From: sender@deer-run.com
To: hal@virtual.com
Subject: testing virtual.com

virtual.com, come in please!
.
250 2.0.0 k057Eni1004247 Message accepted for delivery
mail from: sender@deer-run.com
250 2.1.0 sender@deer-run.com... Sender ok
rcpt to: hal@parallel.com
250 2.1.5 hal@parallel.com... Recipient ok
data
354 Enter mail, end with "." on a line by itself
From: sender@deer-run.com
To: hal@parallel.com
Subject: testing parallel.com

parallel.com, come in please!
.
250 2.0.0 k057Enim004247 Message accepted for delivery
quit
```

You'll notice that I sent both messages via a single SMTP session. You don't have to "quit" and restart between multiple email messages if you don't want to.

The queue IDs we're interested in are "k057Eni1004247" and "k057Enim004247"

7.1.4. Use the queue IDs from the previous exercise to pull out the log entries related to the messages you sent. What do you notice about the recipient addresses in the log entries? Make notes in the space below:

Here are my log entries for the two messages:

```
Jan  4 23:15:54 ext1 sm-mta[4247]: k057Eni1004247:
from=sender@deer-run.com, size=105, class=0, nrcpts=1,
msgid=<200601050715.k057Eni1004247@ext1.sysiphus.com>,
proto=SMTP, daemon=MTA, relay=localhost.localdomain
[127.0.0.1]
Jan  4 23:15:54 ext1 sm-mta[4251]: k057Eni1004247:
to=hal@virtual.com, delay=00:00:32, xdelay=00:00:00,
mailer=smtp, pri=120105, relay=int1.sysiphus.com.
[10.66.10.1], dsn=2.0.0, stat=Sent (k05AiQ5A004097
Message accepted for delivery)
Jan  4 23:16:49 ext1 sm-mta[4247]: k057Enim004247:
from=sender@deer-run.com, size=108, class=0, nrcpts=1,
msgid=<200601050716.k057Enim004247@ext1.sysiphus.com>,
proto=SMTP, daemon=MTA, relay=localhost.localdomain
[127.0.0.1]
Jan  4 23:16:50 ext1 sm-mta[4256]: k057Enim004247:
to=hal@parallel.com, delay=00:00:34, xdelay=00:00:01,
mailer=smtp, pri=120108, relay=int1.sysiphus.com.
[10.66.10.1], dsn=2.0.0, stat=Sent (k05AjOHt004105
Message accepted for delivery)
```

What's interesting to me is that the recipient address ("to=...") in both cases just shows the envelope recipient address of that was entered with the "rcpt to:" command. You don't actually get to see what the `virtusertable` or `domaintable` transforms this recipient address into. I could argue that it might be useful to see this information in addition to the original envelope recipient, but unfortunately that's not the way it works. You'd have to go look in the logs on the next hop mail server to see what recipient address was used when the external relay forwarded the email.

7.1.5. Now take a look at the "virtual-samples" file in your home directory, which contains copies of sample emails sent to the `parallel.com` and `virtual.com` domains. Compare the headers of each email—particularly the "Received:" and "To:" headers. What do you notice about each email? Make notes in the space below.

Here are the relevant headers from the mail sent to `parallel.com`:

```
Received: from intl.sysiphus.com (intl.sysiphus.com [...])
        by elk.sysiphus.com (8.13.4/8.13.4) with ESMTTP id ...
        for <hal@sysiphus.com>; Thu, 5 Jan 2006 18:01:04 ...
Received: from ext1.sysiphus.com (ext1.sysiphus.com [...])
        by intl.sysiphus.com (8.13.4/8.13.4) with ESMTTP id ...
        for <hal@sysiphus.com>; Thu, 5 Jan 2006 02:45:24 -0800
Received: from localhost.localdomain (localhost.localdomain [...])
        by ext1.sysiphus.com (8.13.4/8.13.4) with SMTP id ...
        for hal@parallel.com; Wed, 4 Jan 2006 23:16:16 -0800
[...some headers not shown...]
To: hal@sysiphus.com
```

Notice that after the first (i.e., the lowest) "Received:" header the envelope recipient is transformed ("for <hal@sysiphus.com>") and that the To: address in the headers also gets rewritten to the canonical domain by the `domaintable` feature.

Now here are the headers from the `virtual.com` case:

```
Received: from intl.sysiphus.com (intl.sysiphus.com [...])
        by elk.sysiphus.com (8.13.4/8.13.4) with ESMTTP id ...
        for <marketing@sysiphus.com>; Thu, 5 Jan 2006 18:05:41 ...
Received: from ext1.sysiphus.com (ext1.sysiphus.com [...])
        by intl.sysiphus.com (8.13.4/8.13.4) with ESMTTP id ...
        for <marketing@sysiphus.com>; Thu, 5 Jan 2006 02:48:57 ...
Received: from localhost.localdomain (localhost.localdomain [...])
        by ext1.sysiphus.com (8.13.4/8.13.4) with SMTP id ...
        for hal@virtual.com; Wed, 4 Jan 2006 23:19:26 -0800
[...some headers not shown...]
To: hal@virtual.com
```

Here you'll notice that while the envelope recipient gets rewritten, the `virtusertable` does not rewrite addresses elsewhere in the message headers (like the "To:" address).

Aliases and Mailing Lists



Aliases and Mailing Lists

To use aliases with Sendmail you have to have a mail server that's doing "local delivery", and so far in this course we've been focusing on Sendmail as a relay only. However, it's worth discussing aliases and mailing lists so that you can decide if this functionality is appropriate for your site. After completing this module you should:

- Know how to configure different sorts of aliases in the aliases database
- Set up a mail server where you can define aliases and create mailing lists for your organization
- Know how to deal with routing and addressing issues raised by this configuration
- Understand the basics of mailing list software and how to configure mailing lists



Alias Basics

- Aliases are a simple way to:
 - Forward email to off-site users
 - Create distribution lists
 - Archive certain email into files on disk
 - Trigger "auto-responder" type programs
- Aliases found in `/etc/mail/aliases` (sometimes `/etc/aliases`)
- Must be converted to Unix DB file with `newaliases` (*do not* use `makemap`)

Aliases Overview

An email alias allows administrators to create an email account in their domain that has special behaviors, as opposed to having email to that address simply delivered into a mailbox. Aliases can do a number of different things, including:

- *Forward email to an account in another domain* – This can be useful when one of your users leaves the company and you want to forward their email to their new employer as a courtesy. It's also good when you're operating a "virtual company" and want to forward email from the domain used by the organization to the actual mailboxes where users receive their email (although you can also use the `virtusertable` feature for this).
- *Create distribution lists* – You can create aliases that forward email to more than one user. For example, mail to "softball@sysiphus.com" could go to all the members of the company softball team.
- *Archive email* – You can create aliases that automatically append email to a specified file. Since aliases are allowed to include other aliases, we could add an alias to the end of our "softball@sysiphus.com" distribution list that causes all email sent to this alias to also be archived in a file in case somebody wants to review a previous discussion. The only problem is that the archive file will grow forever without bound unless you also create some external process for periodically starting a new archive file and storing the old archive in some other location. Mailing list software (more on this at the end of this module) generally provides better archiving capabilities than plain aliases.

- *Pipe incoming email into other programs* – Aliases can also be configured to send each incoming email message into an external program. This is good for "auto-responder" type programs: "Send email to `info@sysiphus.com` and we'll send you information about our products". It's also useful when you want to have an email front-end for your customer support system, ordering system, etc. You must be very careful when developing your own programs for processing email from aliases. Appendix B contains some pointers on the "right" way to write these sorts of programs.

The aliases database is just another one of those Unix DB style database files, just like the `mailertable`, `domaintable`, `virtusertable`, and `access DB`. Normally the aliases text file you edit is `/etc/mail/aliases`, although some operating systems use `/etc/aliases` instead. The trick about the aliases database is that you *must not* use `makemap` to rebuild the database. Sendmail comes with a special program called `newaliases` for rebuilding the aliases database (or you can use "`sendmail -bi`" instead). `newaliases` includes some clever functionality for letting the running Sendmail process know when it's in the middle of rebuilding the aliases database so that you don't bounce mail while the database is being rebuilt and is incomplete.



Sample Alias Entries

```
hal: hal@deer-run.com
pomeranz: hal@deer-run.com

staff: hal,laura,jim,staff-archive
staff-archive: /var/archives/staff

postmaster: staff
abuse: staff

info: |/etc/mail/infobot

customers: :include: /lists/customer-list
```

Some Sample Aliases Entries

Notice that the syntax for the aliases file is a little different from the other Sendmail databases we've seen so far. The format of an alias is simply "*<alias>*: *<expansion>*", where *<alias>* is the alias name and *<expansion>* is what you want to happen when people send email to the given alias. Note that the alias name is always unqualified—you never include the domain portion of the email address—because aliases are only expanded if the address is "local" to this machine (in other words, the address is the local machine or one of the domains listed in `local-host-names`).

The examples on the slide demonstrate the correct syntax for different kinds of aliases and illustrate some important points about aliases:

- The first couple of examples demonstrate how to use aliases to forward email to an off-site user. Note that this example also demonstrates how you can use aliases to have multiple addresses that all point to the same user—often useful after one of your users changes their name after getting married.
- The "staff" alias shows how to create an alias that forwards to multiple users—just list all of the recipients separated by commas. You are allowed to use fully-qualified email addresses here if you want. Notice that this alias also references other aliases elsewhere in the file: "hal" and "staff-archive". The various aliases do not need to appear in any special order (they get re-ordered when their stuffed into the DB file by `newaliases` anyway). Remember, however, that if you have aliases that include other aliases, each alias expansion counts as a "hop" on the delivery path

of the message (recall our conversation about the `MAX_HOP` setting earlier in the course).

- The "staff-archive" alias demonstrates the correct syntax for creating an alias that writes email to a file. Sendmail will write these files as an unprivileged user to try and avoid security issues. The rules Sendmail follows get a little complicated here, but the basic idea is that Sendmail will look for any of the following accounts (in order) in the `passwd` file: `mailnull`, `sendmail`, `daemon`. Sendmail will write the file as whatever account it finds, or as `UID 1` if none of the accounts is found. You can also create your own special `UID` and `GID`—let's say we want to use "archive" as the name for our special user and group—and then configure Sendmail to use your `UID` and `GID` with the following macro:

```
define(`confDEF_USER_ID', `archive:archive')
```

- Every aliases file should include "postmaster" and "abuse" aliases that actually forward to a real live human being (or into a trouble ticketing system that gets looked at by a real live human being). The `postmaster` alias is mandated by RFC2822, and the `abuse` alias by RFC2142. Sites often have other common aliases like `webmaster` defined as well.
- The "info" alias demonstrates the correct syntax for creating an alias that pipes mail into a program. Be aware that these kinds of alias cause Sendmail to run the specified program with `root` privileges. So be extremely careful about the kinds of programs you trigger through this mechanism. See Appendix B for further information.
- The "customers" alias is a distribution list that sends email to many different user accounts. But rather than listing those accounts in the aliases file, the email addresses are maintained in an external file that is pulled in using the `:include:` directive. Note that you do not need to rebuild the aliases database each time you update the external file—Sendmail reads the contents of this file every time it receives an email for the specified alias.



When Aliases are Expanded

- Alias expansion is never done by MSP
- MTA must be in "local delivery" mode before it looks at aliases file
- In other words, the RHS of the email destination is in `local-host-names`
- *Problem:* neither our internal nor external relay is doing local delivery

When Alias Expansion Happens

Alias expansion is only done by the MTA process, and never by the MSP. So if you have mail "client" machines where you've disabled the MTA process you cannot have aliases on these systems. You're supposed to use masquerading and forward all of your email traffic up to a central server where you maintain a master aliases file. Maintaining aliases on lots of different machines scattered throughout your network seems like a recipe for confusion to me.

Also, alias expansion only happens during "local" delivery—that is the MTA has looked at its `local-host-names` file and found the domain portion of the recipient email address listed. However, if you have `MAIL_HUB` set the email will be forwarded to the `MAIL_HUB` relay server *before* Sendmail looks at the aliases database on the local machine. In fact, the order of precedence when doing local delivery is:

1. Relay to the `MAIL_HUB` server if set
2. Check the aliases database for a matching entry
3. Look for a `.forward` file* in the user's home directory
4. Deliver the email to the user's mailbox in `/var/mail`

* Users can create `.forward` files in order to route their email to a different email address (many organizations have policies against users forwarding their email off-site, so beware `.forward` files in this case). In addition to simply specifying an alternate email address, users may also pipe their incoming email into an external program—popular choices are the "vacation" program for automatically sending "out of the office" notifications and "procmail" for filtering incoming email into different folders.

We have a small problem now with the hypothetical mail architecture we've been using for our examples. None of our Unix machines running Sendmail are actually doing local delivery! The external mail relays are using `MAIL_HUB` to forward everything to the internal relays. And the `mailertable` on the internal relays is forwarding all of our internal email to different destinations. So what shall we do if we want to use aliases in our internal environment?



What to Do?

- Create a new mail server to handle alias expansions-- "`lists.sysiphus.com`"
- Options for getting mail to this server:
 - Use `mailertable` entry on internal relay
 - Just rely on normal SMTP routing
- First we'll need a `.mc` file for this host
- Then there are a some of tricky issues:
 - "Local" user problems
 - Alias mail from outside
 - Whiny marketing people

Setting Up a Machine to Handle Aliases

Probably the most straightforward approach is to set up a new mail server just for the purpose of doing alias expansions. However, we don't want all of our `sysiphus.com` email going through this machine—just the addresses that actually correspond to aliases.

This is a case where we may not want to do masquerading. Call the new machine "`lists.sysiphus.com`" (most organizations use Unix aliases primarily for mailing lists, so this name makes sense) and have users explicitly send email to "`<alias>@lists.sysiphus.com`" addresses. You could create a special `mailertable` rule on your internal mail relays in order to route email to this machine, but since we're using the literal hostname of the machine on the righthand side of the email address, the normal SMTP routing algorithm will actually get the email to the right place without any further intervention on our part.

We'll obviously need a Sendmail configuration file for the `lists.sysiphus.com` machine, so we'll tackle that on the next slide. However, there are some other problems that crop up once you get the basic configuration in place (like how to let people outside the company send email to your mailing lists—assuming you want that behavior). More on these issues after we deal with the basic configuration of the mail server.

Sample .mc File

```
include(`../m4/cf.m4')
OSTYPE(`linux')

define(`confMIME_FORMAT_ERRORS',`False')
define(`confMAX_HOP',`50')
define(`confTO_IDENT',`0s')
FEATURE(`use_cw_file')
FEATURE(`promiscuous_relay')

SMART_HOST(`internal.sysiphus.com')

MAILER(smtp)
```

The slide shows a basic macro configuration file that's appropriate for the `lists.sysiphus.com` server. It's actually not that different from the configuration we're using on the internal mail relays, with a couple of notable exceptions:

- None of the masquerading options are set.
- We're not bothering with a full `mailertable` configuration. For simple servers like this, `SMART_HOST` is usually sufficient.

While the `MAIL_HUB` directive tells Sendmail to relay local mail to another server, the `SMART_HOST` directive tells Sendmail to send all *non-local* email to some other machine instead of doing the normal SMTP algorithm of delivering the email to the MX server for the recipient's domain. Again, Sendmail decides which email is local vs. non-local depending upon the machine's local host name and the domains that are configured into the `local-host-names` file.

Back in the old days sites would use `SMART_HOST` to forward email via UUCP to a machine that was actually directly connected to the Internet (yes, the first site I administered email for did not actually have an Internet feed). Today, sites sometimes use `SMART_HOST` to forward email through their firewalls to their external relay server(s), because internal machines may be unable to reach the Internet. In our case, however, we're using `SMART_HOST` just to avoid having to have complicated mail routing rules on the `lists.sysiphus.com` machine. Either the email address is for an alias in our local aliases file, or it's mail that's going to some other machine. Let the internal mail relay deal with that stuff.



"Local" User Problem

- Suppose we had an alias like:

```
staff: hal, laura, jim
```

- That would mean forward the email to those users *on this machine*
- The users probably don't have accounts here and we wouldn't want this anyway
- Could do fully-qualified names:

```
staff: hal@sysiphus.com, laura@...
```
- Or if you're lazy, use **LUSER_RELAY**

Dealing With Common Issues

The "Local User" Problem

The first problem you're going to run into on your alias server is with aliases like the "staff" alias you see on the slide. The problem is those unqualified email addresses on the righthand side of the alias. As far as your mail server is concerned, that alias definition says you want email to the `staff` alias to be delivered to the listed users *on the local system*, i.e. `lists.sysiphus.com`. Since those users probably don't even have accounts on the local machine, what you're going to end up doing is bouncing a lot of email. You wouldn't want the email to be delivered locally even if the users did happen to have accounts on this system.

What you probably meant is that you wanted that email to be forwarded to those users at their `sysiphus.com` email addresses. One thing you could do is replace the unqualified email addresses on the righthand side with fully-qualified email addresses: "laura@sysiphus.com" instead of just plain "laura", etc. But it seems like a pain to always have to be typing "@sysiphus.com" after everything in the aliases file. You could also set up individual aliases so that "laura" on the local machine points to "laura@sysiphus.com", but doing that for every user in your company (and maintaining the file when you hire and fire people) just seems like a lot of trouble.

It turns out there's a Sendmail feature, `LUSER_RELAY`, that can help here...



LUSER_RELAY

- **LUSER_RELAY** defines what to do with mail for non-existent users

- In our case:

```
define(`LUSER_RELAY',  
      `smtp:internal.sysiphus.com')
```

- Will still need special aliases for the few users who actually have accounts here:

```
laura: laura@sysiphus.com
```

- Unqualified outgoing recipients qualified with **internal.sysiphus.com**

If Sendmail goes to deliver mail to a local username and doesn't find that user in either the aliases file or the local `passwd` file, the email would normally get bounced. `LUSER_RELAY` tells Sendmail to forward the email to the same username on another system instead of bouncing the email. This is precisely what we want in this case—we use `LUSER_RELAY` to forward this email to the `user@sysiphus.com` addresses for our "missing" users.

The envelope recipient address will get qualified with the name of the system you define as the `LUSER_RELAY`. This is similar to the behavior we saw with `MAIL_HUB` earlier. Luckily we've already set up a `domaintable` entry on our internal relay server to transform the outgoing `user@internal.sysiphus.com` addresses to `user@sysiphus.com` addresses.

One problem that we do have is that our user Laura might actually be an email admin with an account on the `lists.sysiphus.com` machine. So the `LUSER_RELAY` setting won't apply to her, since she actually does have an account in the system `passwd` file. The easiest thing to do here is to create aliases for all of the actual user accounts in the system `passwd` file that simply route local mail for these users to their "`@sysiphus.com`" accounts. There should only be a few user accounts for the system administrators of this machine in any event. Actually, it's possible that those administrative users might even *want* to get local email on this machine—given a choice I'd much rather read my email on a Unix system than from an Exchange server using Outlook (I believe I already mentioned that I'm a "Unix bigot").



Mail From Outside

- Next problem is that outside world has no idea how to get email to this host
- This is a problem if the "**user@lists**" addresses ever escape your company
- Ideally, we'd like the external relay to handle this email properly
- So just add another **mailertable** entry on your external mail relays
- Beware potential DNS issues!

Mail From Outside

If you're purely using the "`@lists.sysiphus.com`" addresses for internal purposes then the configuration we have is fine. But if you ever want to allow external users to reach these aliases, then we're going to need to configure the external relay server(s) to recognize and pass this email. Even if you don't intentionally advertise these addresses outside of your company, one of your users might create an email message that contains both external addresses and one or more of the aliases on the `lists.sysiphus.com` server. When the outside people try to "reply all" they're going to get weird bounce messages.

Actually, some sites prefer not allowing outside email addresses to email to their internal distribution lists, because it helps prevent off-topic traffic and spam. However, the "right" way to deal with this problem ends up being turning your distribution lists into real mailing lists managed by external mailing list software. Not only does real mailing list software allow you to restrict posts to just the members of the list, but it also supports features like list archiving that you probably want for real discussion lists. More on mailing lists in just a second.

Anyway, if you want to let outsiders send email to your `@lists.sysiphus.com` addresses, then just modify the `mailertable` on your external mail relays so that you have a rule to forward this traffic to your internal mail servers:

```
lists.sysiphus.com smtp:internal.sysiphus.com
```


If your firewall rules allowed it, I suppose you could simply forward the email directly to the `lists.sysiphus.com` machine itself, but most sites are fairly restrictive when it comes to allowing SMTP traffic through their firewalls. You'll also need to update your `relay-domains` file so that your external servers will relay email for these addresses.

Once you've got the `mailertable` and the `relay-domains` file configured properly, you can go ahead and add an MX record that directs all traffic for `lists.sysiphus.com` to the external relay server(s). Unfortunately, you've just created an email routing loop. The external relays are going to follow their `mailertable` entry to route this email to the `internal.sysiphus.com` machine. That machine, however, is either going to obey a `mailertable` entry that says to forward this email to the machine `lists.sysiphus.com` or just use the standard SMTP routing rules to find this machine. Either way, the internal mail relay ends up doing the normal SMTP delivery algorithm and looking up the MX record associated with `lists.sysiphus.com` to learn where to forward email for aliases and users on this system. Which means the email is going to go right back out to the external relays, who are going to turn around and fire it right back in to the internal relays. Eventually you reach the `MAX_HOP` limit and the mail bounces.

In the real world, most companies employ what's called a "split-horizon" (sometimes also called "split-brain") DNS configuration. Very simply, what this means is that the company has one set of name servers that provide information about their domain to external sites, and another set of DNS servers that are used by internal systems. Typically you do this for security reasons—the outside world doesn't need to know about most of the hosts on the "inside" of your network, so you hide this information to make things more difficult for attackers. One of the bonuses you get from the split-horizon configuration, however, is that you can have different MX records for the "inside" and "outside" copies of your DNS information. On the "outside" DNS servers, the MX record for `lists.sysiphus.com` would point at `external.sysiphus.com`, but on your "inside" DNS servers you could use `internal.sysiphus.com` instead. Neat.



Whiny Marketing People

- *They want:* "**sales@sysiphus.com**" to be what the outside world uses
- Trick is to enable **virtusertable** on your internal mail relay:
`sales@sysiphus.com sales@lists.sysiphus.com`
- All other **@sysiphus.com** addresses will just fall through to **mailertable**
- You need to put **sysiphus.com** in your **VIRTUAL_DOMAINS_FILE**

Hiding Addressing Complexity

I don't want to seem like I'm unduly ragging on marketing types, but sometimes when you're an email administrator their brilliant schemes do create a lot of extra work for you. Suppose—and this is not an unreasonable request by any means—that they want email to `sales@sysiphus.com` to (a) get stuffed into your CRM system so some sales weenie can follow up on the lead, and (b) generate a friendly auto-responder message with some information about your company. You'd like that email to end up on your `lists.sysiphus.com` machine where you can create an alias with an auto-responder program to deal with this traffic. So the problem is routing `sales@sysiphus.com` email to the address `sales@lists.sysiphus.com`.

It turns out we already know how to route email from an address in one domain to a completely different address—it's called the `virtusertable` feature! What's interesting is that `virtusertable` and `mailertable` "play nice" with each other—addresses get looked up in the `virtusertable` first, and then in the `mailertable` if there was no match found in the `virtusertable`.^{*} So you can add `virtusertable` entries for the `sales@sysiphus.com` address and any other `@sysiphus.com` addresses you want to forward to the `lists.sysiphus.com` machine, and then everything else will just get handled by the `mailertable` and forwarded to the mail servers that handle normal user mail.

^{*} Frankly, I'm not sure if this behavior was intentional on the part of the Sendmail developers or just a useful accident.

However, to get Sendmail to even consult the `virtusertable` for `@sysiphus.com` addresses, you need to configure the `VIRTUAL_DOMAINS_FILE` and put `sysiphus.com` in this file. I realize it seems a little odd to put `sysiphus.com` in this file when it's the canonical email domain for the site and most of the email for the domain will be handled by your `mailertable` rules, but that's how it's got to work.



About Mailing Lists

- Mailing lists are distribution lists where the users themselves can add/drop
 - Typically the administrator is only involved in doing the initial list configuration
 - May be able to delegate this or automate it
- Mailing lists also provide archiving, "digests", and other features
- Unix mailing list software usually works by manipulating `aliases` file

About Mailing Lists

You can certainly use aliases as distribution lists for groups of users, and they work fine for this as long as the group of users is fairly small and the group membership doesn't change that much. Otherwise, alias maintenance starts to be a burden. Also, archiving the traffic on the list with an alias that just appends messages to a file isn't so nice—though I suppose you could write your own program for handling this in a nicer fashion.

Mailing lists allow people to subscribe and unsubscribe themselves from the list at will (although the list admin usually has the option of not allowing users to do either or both of these actions—meaning the list admin would have to add/delete people manually). Mailing list software also typically has built-in list archiving functionality and even "digest" functionality where certain users can elect to receive all of the email messages from the list over a certain span of time (usually the last day worth of messages) in a single large email rather than as individual messages. Mailing lists also typically allow you to do useful things like restrict posting to members of the list only, add a special tag to the "Subject:" line of the message so that list traffic can be filtered easily by users, and even insert special headers or footer text (a disclaimer or unsubscribe instructions) to every message on the mailing list.

The other nice feature about mailing lists is that the email administrators can delegate administration of various mailing lists to other users on a list-by-list basis. So if somebody wants a mailing list for the company softball team, all the administrator has to do is set up the basic infrastructure for the list and then let somebody else manage the list. Usually the process of setting up a new list can be completely automated, and you might even allow users to set up their own mailing lists as needed via a web interface or other tool.

Unix mailing list software normally operates via the aliases file. The addresses users use to subscribe and unsubscribe from the mailing list are typically aliases that invoke the mailing list software, which in turn does add/drops in an external file of email addresses. This file is sometimes read by Sendmail via an `:include:` type alias when mail is actually sent to the mailing list. The point is you typically need a Unix machine capable of doing alias expansion in order to utilize standard Unix mailing list software.



Free Mailing List Software

■ Mailman

- Popular, active development
- Written in Python
- Integrated web front-end

■ Majordomo

- Popular with old-timers, bug fixes only
- Written in Perl
- Separate web front-end called MajorCool

There are two commonly used Open Source mailing list software packages for Unix. Folks who have been setting up mailing lists in the last few years seem to be using the Mailman package (<http://www.gnu.org/software/mailman/>). Mailman is written in the Python scripting language, so you will need the Python interpreter installed on whatever machine is handling your mailing lists. The nice thing about Mailman is that it comes with its own web front-end for list management, searching list archives, and allowing users to subscribe/unsubscribe (although users can also do these operations entirely by email).

Sites that have been on the mailing list bandwagon for a while are probably using Majordomo (<http://www.greatcircle.com/majordomo/>), one of the first mailing list software packages for Unix and still a very useful piece of software. Majordomo is written in Perl, and since most Unix machines these days have a Perl interpreter installed getting Majordomo running is a bit easier. The downside is that Majordomo doesn't have a built-in web interface (in fact, Majordomo really pre-dates the popularization of the web), although there are several available including the popular MajorCool package (<http://www.siliconexus.com/MajorCool/>). Majordomo isn't really being actively maintained anymore other than bug fixes for critical security problems. There was talk of doing Majordomo v2 for a while, but I think that the release of Mailman basically ended that effort.

Rather than go through how to install these packages step-by-step, one of the "hands-on" exercises coming up next will go through the process of configuring Mailman. Appendix A contains information on how to install and configure Majordomo.

Exercises

8.1. Basic Alias Configuration

Exercise Goal – Become familiar with the basics of Unix aliases. We'll also be configuring a mail server that actually does local delivery and thus can expand aliases.

You should find a file called `lists.mc` in your home directory, which is a canned configuration file for our new mail server. Again, this is designed to save you some tedious typing.

8.1.1. Use the `lists.mc` file to produce a new configuration for your test server. Create a modified `submit.mc` file (call it "`submit-lists.mc`") without the masquerading directives (we actually want outgoing email to be qualified with the hostname in this case) and which submits outgoing email to the local MTA at IP address 127.0.0.1. Update the `sendmail.cf` and `submit.cf` files and restart the Sendmail daemons. Write down the commands you use in the space below.

8.1.2. Make a directory called `/var/spool/mail-archives` and make sure that this directory is owned by the "mailnull" user and group. Then create an alias called "buddies-archive" that appends mail to a file called "buddies" in this directory. Also create an alias called "buddies" that sends email to users bob, carol, ted, and alice as well as sending a copy to the buddies-archive alias. Write down the commands and aliases you use in the space below.

Hint: Don't forget to run `newaliases` after making your updates!

8.1.3. Now send a test email to your "buddies" alias. Verify that the archive file is being written properly. It's also interesting to look at the log entries for this alias expansion, so go ahead and take a look at the end of your log file. Take notes in the space below.

8.2. Fun with Mailman

Exercise Goal – Gain some experience setting up Mailman and configuring mailing lists.

The class test servers already have the Mailman software installed, we're just going to focus on configuration issues that arise after the basic software installation. Read the Mailman documentation for information on installing Mailman on your particular platform.

8.2.1 Before even starting to use Mailman you must first create the special "mailman" mailing list that the Mailman software uses to communicate with the site administrators. Run the command `newlist mailman` to create the list. You will be prompted for an email address for the person running the list and a list password—for now just use `root@<yourmachinename>` (i.e., `root@lists1.sysiphus.com` or whatever) as the email address and "mailman" as the password. Follow the additional instructions you receive from the `newlist` program. Take notes on this process in the space below.

Hint: You must add the aliases as instructed by the `newlist` program—cut and paste is your friend! Don't forget "newaliases" afterwards!

8.2.2. The "mailman" mailing list needs special policies applied to it—who's allowed to subscribe, special privacy options, etc. Luckily the Mailman software comes with a special policy file that you can use to set these parameters quickly: Fedora installs this file as `/var/lib/mailman/data/sitelist.cfg`. You apply this policy file with the `config_list` command. Use `config_list -h` to get the help text for this command and see if you can figure out how to load the policy file. Take notes on this process in the space below.

8.2.3. Although most people interact with Mailman via its more user-friendly web interface, there are `add_members`, `remove_members`, and `list_members` command-line programs for manually managing mailing list memberships. If you run these commands with the `-h` option you get some help text that describes how the commands work. See if you can figure out how to subscribe your account on the local machine to the "mailman" mailing list we just created (your email address should be something like `guest01@lists1.sysiphus.com`). Verify that it worked using the `list_members` command. Take notes on this process in the space below.

Hint: If you're entering email addresses interactively, use `<ctrl>-D` when you're done entering addresses.

8.2.4. The next step of the installation process is to install some Mailman-related `cron` jobs and start the Mailman queue runner processes. Happily, the Fedora Linux test systems that we're using in class take care of both of these details via the `/etc/init.d/mailman` boot script. Run this script with the `"start"` argument to install the `cron` jobs and fire up the queue runner processes. Also run the command `"chkconfig mailman on"` to make sure the queue runner processes get started the next time the system reboots. Verify that the queue runner processes are running with the `ps` command and look for the file `/etc/cron.d/mailman` to ensure the `cron` jobs are properly installed. Take notes on this process in the space below.

8.2.5. Next you need to create your site administrator passwords using the `mmsitepass` program. The site administrator is essentially the `"root"` user for the Mailman system. You also have the option of creating a "list administrator" password with `"mmsitepass -c"`—the list administrator is allowed to add and remove mailing lists, but doesn't have full site admin powers. Use these commands to create a site admin password of `"mailman"` and a list admin password of `"list"`. Take notes on this process in the space below.

8.2.6. While you can manually create mailing lists using the `newlist` command and hand-editing your aliases file as we did for the "mailman" mailing list, in most cases you want to use Mailman's web interface for creating new lists. Unfortunately, the web interface lacks the privileges to make updates to the system aliases file. I've written a very simple shell script for monitoring the directory where Mailman maintains its mailing lists and automatically updating the aliases file if new lists are added. You'll find this script in your home directory—it's called "mm-update-aliases". Install the script in `/usr/local/bin` and then use the `crontab` command to create a `cron` job that runs this script every 15 minutes. Take notes on this procedure in the space below.

Hint: You can add new cron jobs with "crontab -e". Use "man 5 crontab" to get more information about the proper syntax for cron entries.

EXPLORE: Now go off and try the Mailman web interface for yourself!

You might first want to try `http://<yourmachine>/mailman/create` to create a new mailing list. You can use either the site admin password or the list admin password in the password field at the bottom of the form.

You can use `http://<yourmachine>/mailman/admin/<listname>` to administer your mailing lists. Try setting some of the different options for the mailing list. Try the "Membership Management" link and use the "Mass Subscription" option to add some addresses to the list of subscribers.

You can also try subscribing yourself by going to the `http://<yourmachine>/mailman/` page and clicking on a list name link. Enter the email address of your account on the local machine (i.e., "guest01@lists1.sysiphus.com" or whatever) so that you can see the confirmation message that new subscribers receive.

Answers to Exercises

8.1. Basic Alias Configuration

8.1.1. Use the `lists.mc` file to produce a new configuration for your test server. Create a modified `submit.mc` file (call it "`submit-lists.mc`") without the masquerading directives (we actually want outgoing email to be qualified with the hostname in this case) and which submits outgoing email to the local MTA at IP address 127.0.0.1. Update the `sendmail.cf` and `submit.cf` files and restart the Sendmail daemons. Write down the commands you use in the space below.

We're back to more or less the original macros in our `submit-lists.mc` file:

```
include(`../m4/cf.m4')
define(`confCF_VERSION', `Submit')
define(`__OSTYPE__', `')
define(`confTIME_ZONE', `USE_TZ')
define(`confDONT_INIT_GROUPS', `True')
FEATURE(`msp', `[127.0.0.1]')
```

Hopefully the configuration update process is old hat by now:

```
$ m4 submit-lists.mc >submit-lists.cf
$ m4 lists.mc > lists.cf
$ /bin/su
Password: <not echoed>
# cp submit-lists.cf /etc/mail/submit.cf
# cp lists.cf /etc/mail/sendmail.cf
# /etc/init.d/sendmail restart
Shutting down sendmail:           [ OK ]
Shutting down sm-client:         [ OK ]
Starting sendmail:               [ OK ]
Starting sm-client:              [ OK ]
```

[Answers to Exercises continue on next page...]

8.1.2. Make a directory called `/var/spool/mail-archives` and make sure that this directory is owned by the "mailnull" user and group. Then create an alias called "buddies-archive" that appends mail to a file called "buddies" in this directory. Also create an alias called "buddies" that sends email to users bob, carol, ted, and alice as well as sending a copy to the buddies-archive alias. Write down the commands and aliases you use in the space below.

Creating the directory should be as simple as:

```
# mkdir /var/spool/mail-archives
# chown mailnull:mailnull /var/spool/mail-archives
```

Your alias entries should look something like (order is unimportant):

```
buddies: bob, carol, ted, alice, buddies-archive
buddies-archive: /var/spool/mail-archives/buddies
```

Once you've created the aliases, it's critical that you run "newaliases" to rebuild your aliases database.

[Answers to Exercises continue on next page...]

8.1.3. Now send a test email to your "buddies" alias. Verify that the archive file is being written properly. It's also interesting to look at the log entries for this alias expansion, so go ahead and take a look at the end of your log file. Take notes in the space below.

Here's the output of the `ls` command on my `mail-archive` directory:

```
-rw----- 1 mailnull mailnull 1254 Jan  5 23:09 buddies
```

As you can see, Sendmail appears to be updating this file properly.

Here are the relevant log entries from the delivery to the "buddies" alias (I just used "echo test message | mail buddies" on the mail server itself to generate the message):

```
Jan  5 23:09:31 lists sendmail[2665]: k0679Vv6002665:
from=root, size=34, class=0, nrcpts=1,
msgid=<200601060709.k0679Vv6002665@lists.sysiphus.com>,
relay=root@localhost
```

```
Jan  5 23:09:31 lists sm-mta[2666]: k0679VWl002666:
from=<root@lists.sysiphus.com>, size=329, class=0,
nrcpts=1, msgid=<200601060709.k0679Vv...@lists.sysiphus.com>,
proto=ESMTP, daemon=MTA, relay=localhost.localdomain
[127.0.0.1]
```

```
Jan  5 23:09:31 lists sendmail[2665]: k0679Vv6002665:
to=buddies, ctladdr=root (0/0), delay=00:00:00,
xdelay=00:00:00, mailer=relay, pri=30034, relay=[127.0.0.1]
[127.0.0.1], dsn=2.0.0, stat=Sent (k0679VWl002666 Message
accepted for delivery)
```

```
Jan  5 23:09:31 lists sm-mta[2667]: k0679VWl002666:
to=/var/spool/mail-archives/buddies, ctladdr=buddies-
archive (47/0), delay=00:00:00, xdelay=00:00:00,
mailer=*file*, pri=150560, dsn=2.0.0, stat=Sent
```

```
Jan  5 23:09:32 lists sm-mta[2667]: k0679VWl002666:
to=bob,carol,ted,alice, ctladdr=<root@lists.sysiphus.com>
(0/0), delay=00:00:01, xdelay=00:00:01, mailer=smtp,
pri=150560, relay=int1.sysiphus.com. [192.168.10.1],
dsn=2.0.0, stat=Sent (k05E0K0T004746 Message accepted for
delivery)
```

You can see the log entries from the MSP process (marked with the program name "sendmail") mixed in with the log entries from the MTA process (mark with "sm-mta"). Notice that the MTA process logs the delivery to the archive file separately from the mail that gets forwarded to the internal relay server for bob, carol, ted, and alice.

8.2. Fun with Mailman

8.2.1 Before even starting to use Mailman you must first create the special "mailman" mailing list that the Mailman software uses to communicate with the site administrators. Run the command "newlist mailman" to create the list. You will be prompted for an email address for the person running the list and a list password—for now just use "root@<yourmachinename>" (i.e., "root@lists1.sysiphus.com" or whatever) as the email address and "mailman" as the password. Follow the additional instructions you receive from the newlist program. Take notes on this process in the space below.

Here's some sample output from the newlist program:

```
# newlist mailman
Email of the person running the list: root@lists.sysiphus.com
Initial mailman password: <not echoed>
To finish creating your mailing list, you must edit your
/etc/aliases (or equivalent) file by adding the following lines,
and possibly running the `newaliases' program:

## mailman mailing list
mailman:          "|/usr/lib/mailman/mail/mailman post mailman"
mailman-admin:   "|/usr/lib/mailman/mail/mailman admin mailman"
[...a whole bunch of other aliases not shown...]

Hit enter to notify mailman owner...
```

So according to the instructions, we have to add all of those aliases. The easiest thing to do here is to just cut and paste the lines above into your aliases file. Don't forget to run newaliases after you do this!

8.2.2. The "mailman" mailing list needs special policies applied to it—who's allowed to subscribe, special privacy options, etc. Luckily the Mailman software comes with a special policy file that you can use to set these parameters quickly: Fedora installs this file as /var/lib/mailman/data/sitelist.cfg. You apply this policy file with the config_list command. Use "config_list -h" to get the help text for this command and see if you can figure out how to load the policy file. Take notes on this process in the space below.

Hopefully you were able to figure out the correct command line:

```
config_list -i \
    /var/lib/mailman/data/sitelist.cfg mailman
```

Note that this policy file is probably not appropriate for normal mailing lists that you create. It's really only intended for the special "mailman" mailing list.

8.2.3. Although most people interact with Mailman via its more user-friendly web interface, there are `add_members`, `remove_members`, and `list_members` command-line programs for manually managing mailing list memberships. If you run these commands with the `-h` option you get some help text that describes how the commands work. See if you can figure out how to subscribe your account on the local machine to the "mailman" mailing list we just created (your email address should be something like "guest01@lists1.sysiphus.com"). Verify that it worked using the `list_members` command. Take notes on this process in the space below.

Normally `add_members` wants to read in email addresses from a file using the `-r <filename>` argument. However, you can get `add_members` to read email addresses from the standard input by using `-` instead of the file name:

```
# add_members -r - mailman
hal@lists.sysiphus.com
^D
Subscribed: hal@lists.sysiphus.com
# list_members mailman
hal@lists.sysiphus.com
```

Use `<ctrl>-D` when you're done entering email addresses and the `add_members` command will subscribe all of the addresses you entered. You can also see the output of the `list_members` command confirming that the address I entered was subscribed.

[Answers to Exercises continue on next page...]

8.2.4. The next step of the installation process is to install some Mailman-related `cron` jobs and start the Mailman queue runner processes. Happily, the Fedora Linux test systems that we're using in class take care of both of these details via the `/etc/init.d/mailman` boot script. Run this script with the "start" argument to install the `cron` jobs and fire up the queue runner processes. Also run the command "`chkconfig mailman on`" to make sure the queue runner processes get started the next time the system reboots. Verify that the queue runner processes are running with the `ps` command and look for the file `/etc/cron.d/mailman` to ensure the `cron` jobs are properly installed. Take notes on this process in the space below.

The commands you run are straightforward:

```
# /etc/init.d/mailman start
Starting mailman:                [ OK ]
# chkconfig mailman on
```

You should be able to run "`ps -ef | grep mailman`" to see the queue runner processes functioning, and "`ls -l /etc/cron.d/mailman`" should confirm that the `cron` jobs were installed. You might want to look at this file and see exactly what sorts of things mailman is doing via `cron`—it's mostly just regular reminders and cleaning up archive files.

8.2.5. Next you need to create your site administrator passwords using the `mmsitepass` program. The site administrator is essentially the "root" user for the Mailman system. You also have the option of creating a "list administrator" password with "`mmsitepass -c`"—the list administrator is allowed to add and remove mailing lists, but doesn't have full site admin powers. Use these commands to create a site admin password of "mailman" and a list admin password of "list". Take notes on this process in the space below.

Again, this is pretty simple:

```
# mmsitepass
New site password: <not echoed>
Again to confirm password: <not echoed>
Password changed.
# mmsitepass -c
New list creator password: <not echoed>
Again to confirm password: <not echoed>
Password changed.
```

Obviously in a real Mailman installation you'd use a much stronger password in each case.

8.2.6. While you can manually create mailing lists using the `newlist` command and hand-editing your aliases file as we did for the "mailman" mailing list, in most cases you want to use Mailman's web interface for creating new lists. Unfortunately, the web interface lacks the privileges to make updates to the system aliases file. I've written a very simple shell script for monitoring the directory where Mailman maintains its mailing lists and automatically updating the aliases file if new lists are added. You'll find this script in your home directory—it's called "mm-update-aliases". Install the script in `/usr/local/bin` and then use the `crontab` command to create a cron job that runs this script every 15 minutes. Take notes on this procedure in the space below.

Assuming you're in the directory where the `mm-update-aliases` script is, installing the script should look something like:

```
# cp mm-update-aliases /usr/local/bin
# chown root:root /usr/local/bin/mm-update-aliases
# chmod 755 /usr/local/bin/mm-update-aliases
```

You can use the "`crontab -e`" command to manually add the cron job. The correct syntax for the cron entry would be:

```
*/15 * * * * /usr/local/bin/mm-update-aliases
```

Note that my approach to solving this problem is a pretty crude hack. David Champion has developed another approach, which you can read about in the file `/usr/share/doc/mailman-*/contrib/mm-handler.readme`.

Wrap Up



Wrap Up

The material in this course certainly isn't everything there is to know about Sendmail, but it's a good start. You need to practice what you've learned for a while before you'll be ready to absorb more, so let's stop here. I've just got some closing remarks and books and URLs for further reading to share. After completing this module you should:

- BE DONE WITH THE COURSE! ☺



Time for Me to Finish Up...

- You have taken the first steps on a long (and sometimes confusing) journey
- Any last questions in real-time?
- Feel free to email me questions you think of in the future...

Any Final Questions?

At best, Sendmail can be described as "sometimes bewildering". Any final questions you have related to the material we've covered today, or any email-related issues you've always wondered about?

My contact information is available on the first page of this course book and I welcome inquiries from former students, and even just random folks on the Internet who've got email questions. I am fairly busy in with my consulting business, so it may take me a few days to respond to your email but I do try to personally respond to each inquiry I get. The exception is when you pose a question that is clearly documented in a well-known book (like the O'Reilly *Sendmail* book) or Internet FAQ. For those kinds of questions, I tend to pick and choose whether I respond or not.

email automatically into different mail folders. For example, `procmail` can recognize the spam score header inserted by Spamassassin and route email that appears to be spam into a separate folder.* Some users use `procmail` to sort their mailing list traffic into different folders automatically. Sendmail even supports using `procmail` instead of the normal "local delivery" mailer so that email admins can configure `procmail` to do something other than just jam new email onto the end of the user's mailbox under `/var/mail`—like perhaps automatically sorting the email into folders in the user's home directory.

- I mentioned that normal SMTP traffic is unencrypted, but Sendmail supports both SSL and Transport Layer Security (TLS) for encrypting SMTP traffic between two servers. You'll need to compile your `sendmail` binary with SSL support for SSL or TLS to function. Sendmail also supports SMTP AUTH(entication) for strongly authenticating remote SMTP sessions. This is most commonly used when you have lots of mobile users that want to use your external mail relay to route their outgoing email when they're on the road. You can't allow just anybody to relay email through your external mail relays or you get blacklisted because you're a promiscuous relay. So you set up your user's mail clients to use SMTP AUTH to authenticate the user at the start of the SMTP session and then configure your external mail relays to allow relaying by authenticated users. SMTP AUTH requires that you compile your Sendmail binary with the Open Source SASL library.
- Sendmail purists will be horrified that I haven't talked at all about Sendmail's internal ruleset language—though I did show you some Sendmail rulesets early on in the course when I showed you that snippet of the internals of the `sendmail.cf` file. Frankly, most of what you need to do with Sendmail you can configure via the `m4` macro language we've been using throughout this course. It's been several years since I've had to write custom Sendmail rulesets for a customer, and I try to avoid it since it makes the site's Sendmail configuration harder to maintain over the long haul. Still, it can be done and it marks you as a "Real Sendmail Expert™". Sendmail also includes a "ruleset testing" mode ("`sendmail -bt`") so that you can test and debug your Sendmail configuration before putting it into production. We were using "`sendmail -v -C ...`" for testing Sendmail configs earlier, but you can also ruleset testing mode for this purpose.

* Actually, if your site isn't currently using Spamassassin to filter spam at your external relays, users can even install Spamassassin in their own home directories and interact with it directly via `procmail` filters. Some people have created very sophisticated anti-spam filters with `procmail`.



Books and URLs

- *Sendmail (3rd Ed)*, Costales and Allman, O'Reilly (1-56592-839-3)
- *Sendmail Performance Tuning*, Christenson, Pearson (0-32111-570-8)
- *Sendmail Milters*, Costales and Flynt, Addison-Wesley (0-32121-333-5)
- *Sendmail 8.13 Companion*, Costales etc, O'Reilly (0-59600-845-7)
- *Sendmail Cookbook*, Craig Hunt, O'Reilly (0-59600-471-0)

Books and URLs for Further Reading

Books

If you administer Sendmail you must own a copy* of the O'Reilly *Sendmail* book—it's called the "bat book" because it has a picture of a flying fox (a large fruit bat) on the cover. The 3rd Edition covers Sendmail v8.12. I'm rather annoyed by the fact that O'Reilly chose to publish a separate book that covers the new features in Sendmail v8.13 rather than just updating the *Sendmail* book—the *Sendmail v8.13 Companion* is almost 200 pages long by itself and there's no way I'm going to lug around both books. The regular *Sendmail* book is good enough for most things, even if you're using Sendmail v8.13.

Nick Christensen's *Sendmail Performance Tuning* is a great book that teaches you a lot about Sendmail's internals in addition to teaching you how to make Sendmail process email really, really fast. Still, you probably don't absolutely need this book unless you're dealing with a fairly large volume of email. If you find yourself needing to write your own Milters, then the Costales and Flynt book is worthwhile. I find myself looking up things in the *Sendmail Cookbook* and then just wishing I had started with the "bat book" in the first place. Still, as a more verbose index into the massive O'Reilly *Sendmail* book, it may be useful for newbies.

* I actually own two copies of this book—one for home and one to carry with me to customer engagements.

URLs

The root of all knowledge for all things related to Sendmail is www.sendmail.org, although the information on this site isn't updated as frequently as I might like. Download the Open Source version of Sendmail from ftp.sendmail.org. Go to www.sendmail.com for information on the commercially-supported version of Sendmail. Lots of information on Sendmail's Milter interface is available at www.milter.org.

There's a decent amount of Sendmail-related information linked off my personal web site, <http://www.deer-run.com/~hal/>. In particular, the two articles I wrote for *Sys Admin Magazine* on MSP configuration issues are a useful supplement to this course:

<http://www.deer-run.com/~hal/sysadmin/sendmail.html>
<http://www.deer-run.com/~hal/sysadmin/sendmail2.html>

Here are some links to different `dnsbls` that you can subscribe to:

<http://www.spamcop.net/>
<http://www.spamhaus.org/>
<http://njabl.org/>
<http://dsbl.org/>
<http://www.ordb.org/>

Here is a bunch of links to different pieces of anti-spam software:

MIMEDefang – <http://www.mimedefang.org/>
MIMEDefang How-To document –
<http://www.mickeyhill.com/mimedefang-howto/>
Spamassassin – <http://spamassassin.apache.org/>
ClamAV – <http://www.clamav.net/>
Vipul's Razor – <http://razor.sourceforge.net/>
Here's what happened to Brightmail –
<http://enterprisesecurity.symantec.com/products/products.cfm?productid=642>
Greylisting (main site) – <http://projects.puremagic.com/greylisting/>
milter-greylis (an Open Source greylisting Milter) –
<http://hcpnet.free.fr/milter-greylis/>

Mailing list software links:

<http://www.gnu.org/software/mailman/>
<http://www.greatcircle.com/majordomo/>
<http://www.siliconexus.com/MajorCool/>

SSL and TLS support require the OpenSSL library – <http://www.openssl.org/>
SMTP AUTH requires the Open Source SASL library – <http://asg.web.cmu.edu/sasl/>

Earlier I stressed the importance of "hardening" the operating system configuration on your mail relays. There are a number of useful guides and tools for helping you with this process, including:

Hardening guides for many operating systems at www.CISecurity.org

A Solaris hardening recipe at <http://www.deer-run.com/~hal/solaris-sbs/>

Hardening tool for Linux, HP-UX, and OS X at www.bastille-linux.org

Hardening tool for Solaris at <http://www.sun.com/software/security/jass/>



There is No More...

Thanks for participating!

Thanks!

Thanks so much for coming. I look forward to seeing you again at a future training event!

Appendix A



Majordomo and Mailing Lists

Appendix A: Majordomo and Mailing Lists

This Appendix covers the basics of setting up Majordomo and configuring it to manage a mailing list. We won't be covering "advanced" functionality like list archiving and digests, but there should be enough material here to get you started with basic list management.



Getting Ready

- Pick a user and group for Majordomo
 - Most sites create a new `majordomo` user
 - OK to run software as group `daemon`
- Pick a local directory for install
- Don't forget a local copy of Perl!

Setting Up Majordomo

Before You Begin

You need to do a little prep work before installing Majordomo. First pick an account for Majordomo to run as. `root` is *not* a good choice. Most sites create a special `majordomo` user with its own unique UID. This is a good choice.

You want to install Majordomo on a non-NFS-mounted directory so your mailing lists don't fail spectacularly when your NFS server dies. Since the Majordomo software is written in Perl, you also need a copy of Perl on your local drive.



Building Majordomo

■ Download software

```
http://www.greatcircle.com  
/majordomo/1.94.5/majordomo-1.94.5.tar.gz
```

■ Unpack and build

```
% zcat majordomo-1.94.5.tar.Z | tar xf -  
% cd majordomo-1.94.5  
% vi Makefile (see next slide)  
% cp sample.cf majordomo.cf  
% vi sample.cf (see notes)  
% make wrapper
```


Download and Compile

Now grab the software from Great Circle and unpack it. You need to edit the `Makefile` as discussed on the next slide.

You also need to copy the `sample.cf` file to `majordomo.cf` and then edit your `majordomo.cf` file. In this file you need to set:

```
$whereami          Your local email domain  
$sendmail_command /usr/sbin/sendmail (or other path?)
```

`make wrapper` actually builds a `suid` binary which is used to safely invoke the Majordomo scripts as the `majordomo` user. Perl has historically had problems with its `suid` support, so the Majordomo developers figured they better write their own wrapper.



Things to Set in `Makefile`

<code>PERL</code>	Location of Perl binary
<code>CC</code>	C compiler
<code>W_HOME</code>	Majordomo install dir
<code>W_USER</code>	UserID for Majordomo
<code>W_GROUP</code>	GroupID for Majordomo
<code>TMPDIR</code>	Place for temp files

These are the variables you need to set in the Majordomo `Makefile`.

Again, your life will be better if your Perl binary is on a local drive and not an NFS directory.

I often install Majordomo in `/etc/mail/majordomo`, but you need a lot of space in your root filesystem for this (`/opt/Majordomo` and `/var/majordomo` are also good choices).

The default `TMPDIR` for Majordomo is `/var/tmp`-- you'll get better performance on some machines (like Solaris systems) if you use `/tmp` when this directory is configured as a RAM disk.



Creating a Mailing List

■ Step 1: Creating initial list file

```
# cd /etc/mail/majordomo/lists
# touch rock-pushers
```

■ Step 2: Create descriptive info file

```
# vi rock-pushers.info
```

Creating a Mailing List

Basic Configuration

In this example, suppose we're creating a mailing list called `rock-pushers@lists.sysiphus.com` alias. Users will be able to send mail to the members of the list by sending mail to this address, and will be able to subscribe/unsubscribe from the list by sending mail to `rock-pushers-request@lists.sysiphus.com`.

The first step is to `cd` into the `.../majordomo/lists` directory (depending on where you installed Majordomo) and create the initial list of recipients for the `rock-pushers` alias. This file can be empty or you can add some initial email addresses to the list right now.

Next you ought to create a file describing the list and its purpose, plus any rules or guidelines for the list and call that file `rock-pushers.info`. This `info` file will be sent to every user when they subscribe and can be requested from the Majordomo server by members of the lists (and outsiders if you permit it) at any time. The `info` file is optional.



Creating a Mailing List (cont.)

■ Step 3: Generate list config file

```
# echo lists | \  
    mailx Majordomo@lists.sysiphus.com  
# vi rock-pushers.config      (see next slide)
```

■ Step 4: Set file ownerships

```
# chmod 644 rock-pushers*  
# chown majordomo rock-pushers*  
# chgrp daemon rock-pushers*
```

Next you need to generate and tweak the `rock-pushers.config` file. The easiest way to generate the `config` file is to have Majordomo do it for you. Unfortunately, Majordomo makes some bad default choices, IMHO. You might want to keep a canonical `config` file around, copy it to the right file name, and tweak it for each new list. We'll talk more about settings in the `config` file on the next slide.

Before you leave the lists directory, be sure to verify that all files are owned by your `majordomo` user and group. The files should be writable by the `majordomo` user and world readable.



Config File Settings

<code>admin_password</code>	<i>default password based on listname</i>
<code>announcements</code>	<i>"no" to turn off informational messages</i>
<code>approve_passwd</code>	<i>default password based on listname</i>
<code>description</code>	<i>fill in something here</i>
<code>index_access</code>	<i>should be "list" just like get_access</i>
<code>message_footer</code>	<i>some put [un]subscribe info here</i>
<code>message_fronter</code>	<i>as above or perhaps disclaimer</i>
<code>message_headers</code>	<i>can set "Errors-To:", etc.</i>
<code>moderate, moderator</code>	<i>if appropriate</i>
<code>subject_prefix</code>	<i>helps people filter list traffic</i>
<code>[un]subscribe_policy</code>	<i>if you care (default is open+confirm)</i>
<code>welcome</code>	<i>useful but turn off if too chatty</i>
<code>which_access</code>	<i>set to "closed" to stop spammers</i>
<code>who_access</code>	<i>ditto</i>

The List Configuration File

There are dozens of settings in the Majordomo list config files. These are just some of the more important ones.

The list passwords end up being a variant of the list name. You probably want to choose better passwords.

Some list managers like to append a disclaimer header/footer and/or instructions on how to unsubscribe from the list. You can also add a special string that is pre-pended to every message subject line to allow people to filter mail more easily.

The default subscribe/unsubscribe policy is `open+confirm`, which means that anybody can subscribe to the list and when they do they have to send back a special confirmation message. This is to avoid people subscribing a person they don't like to every mailing list in the world as a denial of service attack. If you set the subscribe policy to `closed`, then the owner of the list has to approve the subscription.

The `which` command (find out which lists an address is subscribed to) and the `who` command (find out who's on a given list) are often used by spammers to help generate lists of addresses to pester. Shut off access to these commands-- even to list members because spammers can always subscribe to a list.



Creating a Mailing List: Aliases

```
rock-pushers: "|/etc/mail/majordomo/wrapper
               resend -l rock-pushers rock-pushers-outgoing"

rock-pushers-outgoing:
               :include:/etc/mail/majordomo/lists/rock-pushers

rock-pushers-request:
               "|/etc/mail/majordomo/wrapper majordomo
               -l rock-pushers"

rock-pushers-approval: hal
owner-rock-pushers: hal
rock-pushers-owner: hal
```

List-Related Aliases

The first alias invokes the Majordomo `resend` script to forward a message to the recipients of a given list. The `resend` script enforces privacy features like only list members being allowed to send email to the list, as well as appending headers/footers, etc.

The `resend` script just fires the email to another alias which uses the standard alias `include` directive to pull in the contents of the list file corresponding to the `rock-pushers` list in the Majordomo directory. Spammers can completely circumvent the `resend` script and send mail directly to the `outgoing` list and there's nothing you can do about it. Furthermore, the `outgoing` alias appears in the headers of messages sent via the `resend` script, so the alias name is in no way hidden. A fix for this problem would involve hacking Majordomo.

We also need to set up a `rock-pushers-request` alias to allow people to subscribe and unsubscribe from the list. Users can also do this for all lists via the `majordomo` alias. The `request` alias has become a standard for list management, however. You need a real human owner of the list, and if you're moderating the list you need a real human at the `approval` alias. Note that you really only need the `owner-rock-pushers` alias (this is where Majordomo will direct list errors), but the `rock-pushers-owner` form is another one of those de facto standards.

Appendix B



Writing Auto-Responders

Appendix B: Writing Email Auto-Responders

As a mail administrator it helps to be able to write programs that handle mail. You'll often be called on to write auto-responders and even more complicated programs.



Why Cover This?

- Because Sendmail can't do everything for everybody
- Because it can make you look like a serious mail expert
- Because most people get these programs wrong, and it annoys me

Why Should I Learn This?

Sendmail is an extremely complicated program with many features, but it can't do everything. Writing programs to handle mail is a way of extending Sendmail's feature set.

Besides, if you write a cool hack to process mail people will write songs and epics to your fame and people of both sexes will throw themselves at your feet begging to have your love child. Or you might get a bonus, though this is less likely.

Frankly, most people do these sorts of programs badly and end up spamming mailing lists or getting into shouting matches with other auto-responders and causing denial of service attacks. Even worse, badly coded mail programs can allow attackers to get root access on your mail servers and/or destroy data.



40,000 Foot View

Given an alias like

```
aliasname: "|/some/path/program arg1 arg2 ..."
```

- **program** gets text of incoming message on its standard input
- Any arguments are passed as normal
- Programs should exit with status 0 unless there's an error

High-Level Overview

Generally you trigger mail programs out of the aliases file. You can feed command line arguments to the program in the alias file but these arguments are hard-coded into each alias. The email message that triggers the alias is fed into the program on its standard input file descriptor.

Your mail programs should exit with status 0 unless there's a problem. If you exit with an error message postmaster will get an email message like

```
Unknown mailer error: <error message>
```




Our Example

- Used to produce a helpful bounce message when an employee leaves
- One optional argument which is the individual's new mail address
- Want to return the original message to the sender
- Don't want to generate bounce messages to mailing lists, etc.

Our Example

Our example is going to be the standard "this employee no longer works here" auto-responder. You can invoke the alias with

```
john: \  
    "|/etc/mail/natabot john@elsewhere.com"  
mary: "|/etc/mail/natabot"
```

The script takes an optional argument, which is the user's new email address. We want to return the sender's original message back to them and we want to make sure we don't spam mailing lists or respond to other auto-generated mail messages.

The complete source for the `natabot` program is included on the next two pages. The slides that follow we analyze this code bit by bit.

```

#!/usr/bin/perl

# /etc/mail/natabot -- Let people know folks have moved on
#
# Copyright (C) 1998 Hal Pomeranz/Deer Run Associates, hal@deer-run.com
# All rights reserved. Permission to distribute freely under the same
# terms as Perl as long as this copyright and all other comments are
# preserved

# How to use this script:
#
# Assuming user "smith" has changed jobs and is now "smith@other.org",
# just create an alias:
#
#     smith: "|/etc/mail/natabot smith@other.org"
#
# The new address argument is always optional.

$sendmail = '/usr/sbin/sendmail';          # path to Sendmail binary

# Read the first line of the incoming message into @header. Get the
# return address from this first line. Quit silently if we can't
# parse the from line or if message is from MAILER-DAEMON (aka '<>').
# Quit noisily if we detect shell metacharacters in the address.
#
$header[0] = <STDIN>;
($from) = $header[0] =~ /^From\s+(\S+)/;
exit(0)
    if (!length($from) || $from eq '<>' || $from =~ /mailer-daemon/i);
die "Hostile address: $from\n" if ($from =~ /[\\|&];/);

# Now read in the rest of the headers. Quit silently if we find a
# Precedence: header with value junk, list, or bulk. We've reached
# the end of the headers when we hit a blank line.
#
while (<STDIN>) {
    last if (/^$/);
    if (($prec) = /^Precedence:\s+(\S+)/) {
        exit(0) if ($prec =~ /^(junk|list|bulk)$/);
    }
    push(@header, $_);
}

```

```

# OK, time to start sending mail.  Again we try to avoid people messing
# with shell metachars in the $from address by opening a pipe to
# Sendmail the hard way (which doesn't invoke the shell).
#
# Note that we're sending our bounce message from MAILER-DAEMON.
# Theoretically this should mean that any autoresponders that get
# our message will not generate a message of their own.
#
$pid = open(MAIL, "|-");
die "fork() failed: $!\n" unless (defined($pid));
unless ($pid) {          # The child executes this block.
    exec($sendmail, '-f', '<>', $from);
    die "exec() failed: $!\n";
}

# If we get here, we're the parent process.
#
# Print some opening headers (including our own Precedence: header) and
# initial chat into the message.
#
print MAIL <<"EOMesg";
From: MAILER-DAEMON
To: $from
Precedence: junk
Subject: FYI -- Invalid Address

You have sent mail to an address which is no longer valid.
EOMesg

# If we have a new address argument, print that.  Then push the headers
# and the rest of the incoming message into the message we're sending
# back.
#
print MAIL "The individual's new address is: $ARGV[0]\n" if
(length($ARGV[0]));
print MAIL "\nYour message is returned below for your
convenience.\n\n";
print MAIL @header;
print MAIL "\n";
while (<STDIN>) { print MAIL; }

# Close the pipe to the Sendmail program and exit quietly
#
close(MAIL);
exit(0);

```




Sucking in Mail Headers

```
# Scan through headers until blank line.
# If we find a Precedence: header, obey it.
#
while (<STDIN>) {
    last if (/^$/);
    if (($prec) = /^Precedence:\s+(\S+)/) {
        exit(0) if ($prec =~ /^(junk|list|bulk)$/);
    }
    push(@header, $_);
}
```

The Rest of the Headers

Lesson #4 is that the headers are separated from the body of the message by a single blank line (no whitespace, no nothing). You stop processing the headers when you hit this line and treat the rest of the message as an opaque blob of data.

Lesson #5 is that some folks put in a special `Precedence:` header and set the value of the header to `bulk` or `junk` or `list` meaning this message was auto-generated and should not be responded to by auto-responders. While the `Precedence: header` is not an official standard, you may as well obey it if somebody goes to the trouble of adding one.

Lesson #6 is that header lines look like

```
<header>:<whitespace><value>
```

For example:

```
Precedence: bulk
```

<value> may be made up of several words separated by whitespace.

To be completely accurate, headers may continue on multiple lines as long as the continuation lines begin with whitespace. You see this most commonly in `Received:` headers

Received: from ext1.sysiphus.com (ext1.sysiphus.com [...])
by intl.sysiphus.com (8.13.4/8.13.4) with ESMTTP id...
for <marketing@sysiphus.com>; Thu, 5 Jan 2006 ...



Starting Sendmail Safely

```
# Be paranoid of the contents of $from and start
# Sendmail without invoking the shell. Make sure
# we send our message from MAILER-DAEMON to avoid
# autoresponders at remote site.
#
$pid = open(MAIL, "|-");
die "fork() failed: $!\n" unless (defined($pid));
unless ($pid) {
    exec($sendmail, '-f', '<>', $from);
    die "exec() failed: $!\n";
}
```

Invoking Sendmail Very Carefully

Lesson #7 is that you can't be too careful with that potentially forged envelope From address. When you do the normal

```
open(MAIL, "|/usr/lib/sendmail $from") || die...
```

in Perl, Perl is actually invoking `/bin/sh` which then invokes Sendmail. If the From address has embedded shell commands in it, you're in for a world of hurt.

The above gibberish actually fires off a Sendmail process without invoking the shell. It's deep Perl magic, so don't freak if you don't understand what's going on.

The `open(MAIL, "|-")` line causes Perl to `fork()` (make a copy of the current process). The standard input of the new process (generally referred to as the *child*) is the other end of the `MAIL` file handle from the original process (the *parent*). The parent process gets back the process ID of the child while the child gets back 0 from the `open()` -- this is how the parent knows its the parent and the child knows its the child.

The child process then calls `exec()` to fire up Sendmail. `exec()` replaces the child Perl script with the Sendmail program but the Sendmail program inherits the standard input (and all other file descriptors) from the original child Perl script.

The parent just proceeds on as normal as if it had done the standard `open()` call.



Creating Bounce Message (1)

```
# Create our own mail headers (including
# Precedence:) and introductory chat.
#
print MAIL <<"EOMesg";
From: MAILER-DAEMON
To: $from
Precedence: junk
Subject: FYI -- Invalid Address

You have sent mail to an invalid address.
EOMesg
```

Creating Our Outgoing Email Message

Lesson #8 is how to create email messages on the fly.

The parent process now starts composing a mail message and feeding it to the Sendmail process. We create a few headers (including our own `Precedence:` header) and then a blank line to separate the headers from the message body (that's *Lesson #4*).



Creating Bounce Message (2)

```
# Add new address if we have one.
# Include headers read so far and rest of msg.
#
print MAIL "Use new address: $ARGV[0]\n"
    if (length($ARGV[0]));
print MAIL "\nYour message returned below.\n\n";
print MAIL @header;
print MAIL "\n";
while (<STDIN>) { print MAIL; }
```

We add a line with the user's new address (if any). And then we include the original message (headers first, then the rest of the message body text) to send back to the sender of the original mail.

Lesson #9 is that it's always polite to return the original message sent to an auto-responder. Not everybody configures their mail program to keep a copy of all messages they send out (although they should!).



Finishing Up

```
# Close the pipe to the Sendmail program.  
# Exit quietly.  
#  
close(MAIL);  
exit(0);
```

Finishing Up

Finally we close the `MAIL` file descriptor and this causes the Sendmail program to fire off our response and exit. Then we exit ourselves with exit status 0 meaning that everything is OK.

Lesson #10 is always clean up after yourself.

You knew there were going to be 10 lessons, didn't you?