```
#!/usr/bin/env python3
import os
import sys
import subprocess
import logging
from pathlib import Path
# Set up logging to file and console
logging.basicConfig(
level=logging.INFO,
format="%(asctime)s [%(levelname)s] %(message)s",
handlers=[
logging.FileHandler("boot_repair.log"),
logging.StreamHandler(sys.stdout)
]
)
def run_cmd(cmd, check=False):
"""Run a shell command and return (exit_code, output). Log errors if any."""
logging.debug(f"Running command: {' '.join(cmd)}")
try:
result = subprocess.run(cmd, capture_output=True, text=True, check=check)
output = result.stdout.strip()
if result.stderr:
logging.debug(result.stderr.strip())
return result.returncode, output
except subprocess.CalledProcessError as e:
```

logging.error(f"Command {' '.join(cmd)} failed with code {e.returncode}: {e.stderr}")

return e.returncode, e.stderr

def find_windows_partition():

"""Find the partition that contains the Windows installation by looking for an NTFS partition with a Windows

folder."""

logging.info("Scanning for Windows installations (NTFS partitions)...")

Use lsblk to get partition info in JSON for easier parsing

code, lsblk_json = run_cmd(["lsblk", "-J", "-b", "-o", "NAME,FSTYPE,SIZE"], check=False)

if code != 0:

logging.error("Failed to list block devices. Ensure lsblk is available.")

return None

import json

try:

data = json.loads(lsblk_json)

except json.JSONDecodeError:

logging.error("Could not parse lsblk output.")

return None

windows_partition = None

windows_size = 0

for device in data.get("blockdevices", []):

lsblk lists disks and their partitions under "children"

children = device.get("children", [])

if not children:

continue

```
for part in children:
fstype = part.get("fstype")
if fstype and fstype.lower() == "ntfs":
part_name = part.get("name")
part_size = int(part.get("size", 0))
dev_path = f"/dev/{part_name}"
logging.debug(f"Found NTFS partition: {dev_path}, size {part_size} bytes")
# Mount the partition to check if it's the OS (contains Windows folder)
mount_point = Path("/mnt/windows")
mount_point.mkdir(exist_ok=True)
# Mount read-only for safety
run_cmd(["umount", str(mount_point)]) # ensure not already mounted
ret, _ = run_cmd(["mount", "-o", "ro", dev_path, str(mount_point)])
if ret != 0:
logging.debug(f"Could not mount {dev_path} (maybe in use or need root). Skipping.")
continue
# Check for Windows folder
if (mount_point / "Windows" / "System32").exists():
logging.info(f"Windows installation found on {dev_path}")
windows_partition = dev_path
windows_size = part_size
# Keep it mounted for later use (we will unmount at end or as needed)
# We break out after finding the first (assuming one Windows OS).
# If multiple Windows installations exist, this picks the first found.
break
# Not the OS partition, unmount and continue
```

```
run_cmd(["umount", str(mount_point)])
if windows_partition:
break
if not windows_partition:
logging.error("No Windows installation (NTFS partition with Windows folder) was found.")
return windows_partition
def detect_boot_mode(disk):
"""Detect if the Windows disk is GPT/UEFI or MBR/BIOS by checking partition table type."""
# Use blkid to get partition table type (PTTYPE)
code, pt_type = run_cmd(["blkid", "-s", "PTTYPE", "-o", "value", disk])
if code != 0 or not pt_type:
logging.warning("Could not determine partition table type. Assuming BIOS/MBR.")
return "BIOS"
pt_type = pt_type.strip().lower()
if pt_type == "gpt":
logging.info(f"Disk {disk} has GPT partition table: UEFI mode assumed.")
return "UEFI"
elif pt_type == "dos":
logging.info(f"Disk {disk} has DOS/MBR partition table: Legacy BIOS mode assumed.")
return "BIOS"
else:
# Unexpected type, default to BIOS for safety
logging.warning(f"Unknown partition table type '{pt_type}'. Assuming BIOS.")
return "BIOS"
def get_efi_partition(disk):
```

"""Find the EFI System Partition (ESP) on the given disk (for UEFI systems)."""

Use blkid to find a FAT32 partition labeled as EFI (if available)

code, blkid_out = run_cmd(["blkid"], check=False)

if code != 0:

return None

efi_part = None

for line in blkid_out.splitlines():

Typical EFI partition: TYPE="vfat" and perhaps PARTLABEL="EFI System"

if disk in line and 'TYPE="vfat"' in line:

This is a FAT partition on the same disk

Further ensure it's likely the ESP by size or label

if 'PARTLABEL="EFI System"' in line or 'PARTLABEL="EFI system partition"' in line or 'EFI' in line:

Extract the device name at start of line

dev_name = line.split(":")[0]

efi_part = dev_name

break

if efi_part:

logging.info(f"EFI System Partition found: {efi_part}")

else:

logging.error("EFI System Partition not found on disk. UEFI boot repair cannot proceed.")

return efi_part

def mount_partition(dev, mount_point):

"""Mount a partition to the given mount point (creates mount_point if not exist)."""

Path(mount_point).mkdir(parents=True, exist_ok=True)

Try to mount (read-write by default)

```
ret, _ = run_cmd(["mount", dev, mount_point])

if ret != 0:

logging.error(f"Failed to mount {dev} on {mount_point}.")

return False

logging.debug(f"Mounted {dev} at {mount_point}.")

return True

def ensure_boot_files_uefi(windows_dev, efi_dev):

"""Ensure that essential Windows boot files exist on the EFI partition. Copy from Windows partition if needed."""

efi_mount = "/mnt/efi"

win_mount = "/mnt/windows"

# Mount the EFI partition

if not mount_partition(efi_dev, efi_mount):

return False

# Ensure Windows partition is mounted (it might already be at /mnt/windows from earlier, mount again if not)

if not Path(win_mount).is_mount():

if not mount_partition(windows_dev, win_mount):

run_cmd(["umount", efi_mount])

return False

bootmgfw_path = Path(efi_mount) / "EFI" / "Microsoft" / "Boot" / "bootmgfw.efi"

bcd_path = Path(efi_mount) / "EFI" / "Microsoft" / "Boot" / "BCD"

need_copy = False

if not bootmgfw_path.exists():

logging.warning("Windows Boot Manager file (bootmgfw.efi) is missing from EFI partition.")

# Check if it exists in the Windows partition

source_path = Path(win_mount) / "Windows" / "Boot" / "EFI" / "bootmgfw.efi"
```

```
if source_path.exists():
# Prompt user for confirmation to copy the file
user_input = input("Windows boot manager not found on EFI partition. Copy it from Windows drive? (y/N):
").strip().lower()
if user_input == 'y':
# Make the target directory if needed
os.makedirs(bootmgfw_path.parent, exist_ok=True)
try:
# Copy file
with open(source_path, "rb") as src, open(bootmgfw_path, "wb") as dst:
dst.write(src.read())
logging.info(f"Copied {source_path} to {bootmgfw_path}")
need_copy = True
except Exception as e:
logging.error(f"Failed to copy bootmgfw.efi: {e}")
else:
logging.info("Skipping copying bootmgfw.efi. UEFI boot might fail without it.")
else:
logging.error("bootmgfw.efi not found in Windows partition. Cannot restore automatically.")
else:
logging.info("Windows boot manager file is present on the EFI partition.")
if not bcd_path.exists():
logging.warning("BCD store is missing from EFI partition. Windows may not boot without a BCD.")
```

```
logging.warning("You may need to run Windows recovery (bcdboot or bootrec) to recreate BCD.")
else:
logging.info("BCD store exists on the EFI partition.")
# Optionally ensure fallback bootloader (bootx64.efi) is present
bootx64_path = Path(efi_mount) / "EFI" / "Boot" / "bootx64.efi"
if bootmgfw_path.exists():
if not bootx64_path.exists():
# Copy bootmgfw.efi to bootx64.efi as a fallback (common practice for safety)
try:
os.makedirs(bootx64_path.parent, exist_ok=True)
with open(bootmgfw_path, "rb") as src, open(bootx64_path, "wb") as dst:
dst.write(src.read())
logging.info("Copied bootmgfw.efi to EFI/Boot/bootx64.efi as fallback boot loader.")
except Exception as e:
logging.error(f"Failed to create fallback bootx64.efi: {e}")
else:
logging.debug("Fallback bootx64.efi already exists on EFI partition.")
# Keep EFI mounted for efibootmgr step (will unmount later)
return True
def repair_uefi_boot(windows_dev, disk):
"""Perform UEFI boot repair: ensure files and create UEFI NVRAM entry."""
# Find and mount EFI partition
efi_dev = get_efi_partition(disk)
if not efi_dev:
return False
```

if not ensure_boot_files_uefi(windows_dev, efi_dev):

If we couldn't ensure boot files, abort UEFI repair

run_cmd(["umount", "/mnt/efi"])

run_cmd(["umount", "/mnt/windows"])

return False

Check if we're in a UEFI-booted environment

if not Path("/sys/firmware/efi").exists():

logging.warning("Live environment is not booted in UEFI mode; cannot run efibootmgr to update NVRAM.")

print("WARNING: Current environment is not UEFI. Skipping efibootmgr step. You may need to add the boot

entry manually in UEFI firmware.")

Unmount and finish

run_cmd(["umount", "/mnt/efi"])

run_cmd(["umount", "/mnt/windows"])

return True

Ensure efibootmgr is installed

code, _ = run_cmd(["which", "efibootmgr"])

if code != 0:

user_input = input("efibootmgr not found. Install it now? (y/N): ").strip().lower()

if user_input == 'y':

run_cmd(["apt-get", "update"])

ret_code, _ = run_cmd(["apt-get", "-y", "install", "efibootmgr"])

if ret_code != 0:

logging.error("Failed to install efibootmgr. Cannot modify UEFI boot entries.")

Unmount and exit

```
run_cmd(["umount", "/mnt/efi"])
run_cmd(["umount", "/mnt/windows"])
return False
else:
logging.info("Skipping UEFI boot entry creation since efibootmgr is not installed.")
run_cmd(["umount", "/mnt/efi"])
run_cmd(["umount", "/mnt/windows"])
return True
# Create Windows Boot Manager entry in UEFI
logging.info("Creating UEFI boot entry for Windows...")
# Determine partition number of the EFI partition (e.g., /dev/sda2 -> 2)
part_num = ''.join(filter(str.isdigit, efi_dev.split('/')[-1]))
# Form efibootmgr command:
cmd = [
"efibootmgr", "-c",
"-d", disk,
"-p", part_num,
"-L", "Windows Boot Manager",
"-l", "\\EFI\\Microsoft\\Boot\\bootmgfw.efi"
]
ret, out = run_cmd(cmd)
if ret == 0:
logging.info("Windows Boot Manager entry created/updated successfully.")
else:
logging.error(f"efibootmgr failed to create boot entry. Output:\n{out}")
logging.error("You may need to add the UEFI boot entry manually via firmware settings.")
```

Unmount EFI and Windows partitions

run_cmd(["umount", "/mnt/efi"])

run_cmd(["umount", "/mnt/windows"])

return True

def ensure_boot_files_bios(windows_dev):

"""Ensure bootmgr and BCD exist on the Windows (or system reserved) partition for BIOS boot."""

boot_mount = "/mnt/bootpart"

Mount the partition (could be the same as Windows partition or a separate system partition)

if not mount_partition(windows_dev, boot_mount):

return False

bootmgr_path = Path(boot_mount) / "bootmgr"

bcd_path = Path(boot_mount) / "Boot" / "BCD"

if bootmgr_path.exists():

logging.info("Found bootmgr on the active partition.")

else:

logging.error("bootmgr is missing from the active partition.")

logging.error("You may need to run Windows recovery (bootrec /fixboot) to fix the boot sector.")

if bcd_path.exists():

logging.info("Found BCD store on the active partition.")

else:

logging.error("BCD store is missing from the active partition.")

logging.error("You may need to rebuild BCD using Windows recovery tools.")

run_cmd(["umount", boot_mount])

```
return True
def set_active_partition(disk, part_num):
"""Mark the given partition as active (boot flag on) on an MBR disk."""
logging.info(f"Setting partition {part_num} on {disk} as the active (bootable) partition.")
ret, _ = run_cmd(["parted", "-s", disk, "set", str(part_num), "boot", "on"])
if ret != 0:
logging.error(f"Failed to set partition {part_num} as active.")
# In MBR, only one partition can be active; parted should handle toggling others off.
def repair_bios_boot(windows_dev, disk):
"""Perform BIOS boot repair: set active partition, ensure files, write MBR."""
# Determine the partition number of the Windows partition
part_num = ''.join(filter(str.isdigit, windows_dev.split('/')[-1]))
# Check current active partition using fdisk
code, fdisk_out = run_cmd(["fdisk", "-l", disk])
active_part = None
if code == 0:
for line in fdisk_out.splitlines():
# In fdisk output, active partition has a '*' in the Boot column
if line.strip().startswith(disk) and "*" in line:
# e.g., "/dev/sda1 *"
parts = line.split()
if parts[0].startswith(disk):
active_part = parts[0]
break
if active_part:
logging.info(f"Current active (boot) partition is {active_part}.")
```

else:

logging.info("No partition is currently marked active.")

If the current active partition is not the Windows partition, or none is active, set the Windows partition active

win_part_path = windows_dev # e.g., /dev/sda1, etc.

if not active_part or active_part != win_part_path:

user_input = input(f"Mark {win_part_path} as the active boot partition? (y/N): ").strip().lower()

if user_input == 'y':

set_active_partition(disk, part_num)

else:

logging.warning("Active partition not changed. BIOS boot may fail if the wrong partition is active.")

Ensure boot files (bootmgr/BCD) on that partition

ensure_boot_files_bios(win_part_path)

Write MBR boot code to the disk

user_input = input(f"Write Windows MBR boot code to {disk}? This will overwrite the MBR. (y/N):

").strip().lower()

if user_input == 'y':

Ensure syslinux's MBR binary is available

mbr_bin = "/usr/lib/syslinux/mbr.bin"

if not Path(mbr_bin).exists():

logging.info("syslinux not found. Installing syslinux to obtain MBR binary...")

run_cmd(["apt-get", "update"])

run_cmd(["apt-get", "-y", "install", "syslinux"])

```
if Path(mbr_bin).exists():

# Use dd to write MBR boot code (without altering partition table). Copy first 440 bytes (to preserve disk

signature and partition table).

ret, _ = run_cmd(["dd", f"if={mbr_bin}", f"of={disk}", "bs=440", "count=1", "conv=notrunc"])

if ret == 0:

logging.info(f"Standard MBR boot code written to {disk}&#8203;:contentReference[oaicite:4]{index=4}.")

else:

logging.error("Failed to write MBR boot code. You may need to run this step manually.")

else:

logging.error("MBR binary not found. Unable to write MBR. Install syslinux or use Windows recovery to fix

MBR.")

else:

logging.info("Skipped writing MBR boot code.")

return True

def main():

logging.info("Starting Windows boot repair script...")

windows_part = find_windows_partition()

if not windows_part:

print("Windows installation not found. Exiting.")

return 1

# Determine the disk containing the Windows partition (e.g., /dev/sda from /dev/sda4)

disk = windows_part

# Strip partition number to get disk (handles /dev/sda1, /dev/nvme0n1p2, etc.)

if disk.endswith(("p", "P")):
```

This covers cases like nvme0n1p2 (ends with p2? Actually nvme0n1p2 doesn't end with 'p', it contains 'p')

We can specifically handle NVMe: partition device ends with 'px' where x is part number.

But simpler: check common patterns:

If device name contains 'nvme' and 'p', then strip up to 'p[0-9]'

import re

disk = re.sub(r"p[0-9]+$", "", disk)

else:

For standard devices like /dev/sda1, strip last digit(s)

disk = ''.join(filter(lambda x: not x.isdigit(), disk))

logging.info(f"Windows partition is {windows_part} on disk {disk}")

mode = detect_boot_mode(disk)

if mode == "UEFI":

print(f"Detected UEFI/GPT system. Proceeding with UEFI boot repair on {disk}...")

repair_uefi_boot(windows_part, disk)

else:

print(f"Detected BIOS/Legacy system. Proceeding with BIOS boot repair on {disk}...")

repair_bios_boot(windows_part, disk)

logging.info("Windows boot repair script completed.")

print("Boot repair process completed. Please review any above messages or boot_repair.log for details.")

if __name__ == "__main__":

try:

main()

except Exception as e:

logging.exception(f"An unexpected error occurred: {e}")

```
print("An error occurred. Please check the log file for details.")
```