Rocky.py

```
#!/usr/bin/env python3

import subprocess
import csv
import os
import re
import stat
import glob
import platform
import datetime
import shutil
import pwd
import grp
import csv

# [U-01] 1.1 root 계정 원격접속 제한 점검
def check_u01():
    ssh_config = "/etc/ssh/sshd_config"

    if not os.path.isfile(ssh_config):
        return "U-01,양호,SSH 서비스가 설치되어 있지 않거나 설정 파일이 존재하지
않음"

    try:
        with open(ssh_config, "r") as f:
            for line in f:
                line = line.strip()
                if line.lower().startswith("permitrootlogin"):
                    value = line.split()[1].lower()
                    if value == "no":
                        return "U-01,양호,root 계정의 원격 접속이 제한됨 (PermitRootLo
gin no)"
                    else:
                        return "U-01,취약,root 계정의 원격 접속이 허용됨 (PermitRootLo
```

```
gin 설정 필요)"
        # 설정이 아예 없는 경우 → 기본값이 yes
        return "U-01,취약,PermitRootLogin 설정이 존재하지 않아 root 계정 접속이
허용될 수 있음"
    except Exception:
        return "U-01,오류,sshd_config 파일을 읽는 중 오류 발생"


#1.2 패스워드 복잡성 설정
def check_u02():
    pw_config = "/etc/security/pwquality.conf"
    if not os.path.isfile(pw_config):
        return "U-02,취약,pwquality.conf 파일이 존재하지 않음"

    def grep_contains(pattern):
        try:
            result = subprocess.run(["grep", "-Ei", pattern, pw_config],
                                    stdout=subprocess.DEVNULL,
                                    stderr=subprocess.DEVNULL)
            return result.returncode == 0
        except Exception:
            return False

    conditions = [
        grep_contains(r"minlen\s*=\s*[8-9]|minlen\s*=\s*[1-9][0-9]+"),
        grep_contains(r"dcredit\s*=\s*-\d+"),
        grep_contains(r"ucredit\s*=\s*-\d+"),
        grep_contains(r"lcredit\s*=\s*-\d+"),
        grep_contains(r"ocredit\s*=\s*-\d+"),
    ]

    if all(conditions):
        return "U-02,양호,패스워드 복잡성 설정이 적절히 구성됨"
    else:
        return "U-02,취약,패스워드 복잡성 설정이 미흡함 (pwquality.conf 설정 필
요)"
```

```
# [U-03] 1.3 계정 잠금 임계값 설정
def check_u03():
    pam_file = "/etc/pam.d/system-auth"
    required_deny = 5
    required_unlock_time = 600

    if not os.path.exists(pam_file):
        return "U-03,취약,system-auth 파일이 존재하지 않음"

    try:
        with open(pam_file, "r") as f:
            lines = [line.strip() for line in f if not line.strip().startswith("#")]

        deny, unlock_time = None, None
        for line in lines:
            if "pam_faillock.so" in line:
                if "deny=" in line and deny is None:
                    match = re.search(r"deny=(\d+)", line)
                    if match:
                        deny = int(match.group(1))
                if "unlock_time=" in line and unlock_time is None:
                    match = re.search(r"unlock_time=(\d+)", line)
                    if match:
                        unlock_time = int(match.group(1))

        if deny is None or unlock_time is None:
            return "U-03,취약,계정 잠금 임계값 설정 미흡"
        if deny >= required_deny and unlock_time >= required_unlock_time:
            return "U-03,양호,계정 잠금 임계값 적절히 설정됨"
        else:
            return "U-03,취약,계정 잠금 임계값 설정 미흡"

    except Exception as e:
        return f"U-03,오류,파일 처리 중 오류 발생: {str(e)}"


# [U-04] 1.4 패스워드 파일 보호
def check_u04():
```

```
    passwd_file = "/etc/passwd"
    shadow_file = "/etc/shadow"

    if not os.path.exists(passwd_file) or not os.path.exists(shadow_file):
        return "U-04,취약,패스워드 파일 존재하지 않음"

    try:
        passwd_perm = os.stat(passwd_file).st_mode & 0o777
        shadow_perm = os.stat(shadow_file).st_mode & 0o777

        if passwd_perm <= 0o644 and shadow_perm <= 0o640:
            return "U-04,양호,패스워드 파일 권한 적절함"
        else:
            return "U-04,취약,패스워드 파일 권한 부적절"
    except Exception as e:
        return f"U-04,오류,권한 확인 중 오류 발생: {str(e)}"


# [U-05] 1.5 root 홈, 패스 디렉터리 권한 및 PATH 설정
def check_u05():
    path_value = os.environ.get("PATH", "")
    is_vulnerable = False

    # '.'이 앞이나 중간에 있는 경우
    if path_value.startswith(".:") or ":.:" in path_value:
        return "U-05,취약,PATH 환경변수에 '.'이 앞이나 중간에 포함됨"

    # 빈 경로(::) 포함 여부
    if "::" in path_value:
        return "U-05,취약,PATH 환경변수에 빈 경로(::) 포함"

    return "U-05,양호,PATH 환경변수에 '.' 또는 빈 경로(::)가 포함되지 않음"


# [U-06] 1.6 파일 및 디렉터리 소유자 설정
def check_u06():
    check_paths = ["/etc", "/var", "/home", "/usr"]
    nouser_files = []
```

```
    nogroup_files = []

    for path in check_paths:
        try:
            result = subprocess.run(["find", path, "-nouser"], stdout=subproces
s.PIPE, stderr=subprocess.DEVNULL, text=True)
            nouser_files.extend(result.stdout.strip().splitlines())

            result = subprocess.run(["find", path, "-nogroup"], stdout=subproce
ss.PIPE, stderr=subprocess.DEVNULL, text=True)
            nogroup_files.extend(result.stdout.strip().splitlines())
        except Exception as e:
            return f"U-06,오류,{path} 경로 점검 중 오류 발생: {str(e)}"

    if not nouser_files and not nogroup_files:
        return "U-06,양호,소유자 및 그룹이 없는 파일이나 디렉터리가 없습니다."
    else:
        messages = ["U-06,취약,소유자 또는 그룹이 없는 파일/디렉터리가 발견되었
습니다."]
        if nouser_files:
            messages.append("\n소유자 없는 파일 및 디렉터리:")
            messages.extend(nouser_files)
        if nogroup_files:
            messages.append("\n그룹 없는 파일 및 디렉터리:")
            messages.extend(nogroup_files)
        return "\n".join(messages)


# [U-07] 1.7 /etc/passwd 파일 소유자 및 권한 설정
def check_u07():
    file_path = "/etc/passwd"

    try:
        stat_info = os.stat(file_path)
        owner = pwd.getpwuid(stat_info.st_uid).pw_name
        group = grp.getgrgid(stat_info.st_gid).gr_name
        perm = stat.S_IMODE(stat_info.st_mode)
```

```
        if owner != "root" or group != "root" or perm > 0o644:
            return "U-07,취약,/etc/passwd 파일 소유자/그룹/권한 부적절"
        else:
            return "U-07,양호,/etc/passwd 파일 소유자/그룹/권한 적절"
    except Exception as e:
        return f"U-07,오류,/etc/passwd 파일 점검 중 오류 발생: {str(e)}"


# [U-08] 1.8 /etc/shadow 파일 소유자 및 권한 설정
def check_u08():
    file_path = "/etc/shadow"

    if not os.path.exists(file_path):
        return "U-08,취약,/etc/shadow 파일이 존재하지 않음"

    try:
        stat_info = os.stat(file_path)
        owner = pwd.getpwuid(stat_info.st_uid).pw_name
        group = grp.getgrgid(stat_info.st_gid).gr_name
        perm = stat.S_IMODE(stat_info.st_mode)

        if owner == "root" and group == "root" and perm in (0o000, 0o400):
            return "U-08,양호,/etc/shadow 파일 소유자 및 권한 설정 적절"
        else:
            return "U-08,취약,/etc/shadow 파일 소유자 및 권한 설정 미흡"
    except Exception as e:
        return f"U-08,오류,/etc/shadow 파일 점검 중 오류 발생: {str(e)}"


# [U-09] 1.9 /etc/hosts 파일 소유자 및 권한 설정
def check_u09():
    file_path = "/etc/hosts"

    if not os.path.exists(file_path):
        return "U-09,취약,/etc/hosts 파일이 존재하지 않음"

    try:
        stat_info = os.stat(file_path)
```

```
        owner = pwd.getpwuid(stat_info.st_uid).pw_name
        group = grp.getgrgid(stat_info.st_gid).gr_name
        perm = stat.S_IMODE(stat_info.st_mode)

        if owner == "root" and group == "root" and perm <= 0o600:
            return "U-09,양호,/etc/hosts 파일 소유자 및 권한 설정 적절"
        else:
            return "U-09,취약,/etc/hosts 파일 소유자 및 권한 설정 미흡"
    except Exception as e:
        return f"U-09,오류,/etc/hosts 파일 점검 중 오류 발생: {str(e)}"


# [U-10] 1.10 /etc/(x)inetd.conf 파일 소유자 및 권한 설정
def check_u10():
    files = ["/etc/inetd.conf", "/etc/xinetd.conf"]
    found = False
    results = []

    for file in files:
        if os.path.exists(file):
            found = True
            try:
                stat_info = os.stat(file)
                owner = pwd.getpwuid(stat_info.st_uid).pw_name
                group = grp.getgrgid(stat_info.st_gid).gr_name
                perm = stat.S_IMODE(stat_info.st_mode)

                if owner == "root" and group == "root" and perm <= 0o600:
                    results.append(f"U-10,양호,{file} 소유자 및 권한 설정 적절")
                else:
                    results.append(f"U-10,취약,{file} 소유자 및 권한 설정 미흡")
            except Exception as e:
                results.append(f"U-10,오류,{file} 점검 중 오류 발생: {str(e)}")

    if not found:
        results.append("U-10,양호,/etc/(x)inetd.conf 파일 존재하지 않음")

    return "\n".join(results)
```

```
# [U-11] 2.7 /etc/syslog.conf 파일 소유자 및 권한 설정
def check_u11():
    files = ["/etc/syslog.conf"]
    found = False
    results = []

    for file in files:
        if os.path.exists(file):
            found = True
            stat_info = os.stat(file)
            owner = pwd.getpwuid(stat_info.st_uid).pw_name
            group = grp.getgrgid(stat_info.st_gid).gr_name
            perm = stat.S_IMODE(stat_info.st_mode)

            if owner == "root" and group == "root" and perm <= 0o640:
                results.append(f"U-11,양호,{file} 소유자 및 권한 설정 적절")
            else:
                results.append(f"U-11,취약,{file} 소유자 및 권한 설정 미흡")

    if not found:
        results.append("U-11,양호,/etc/syslog.conf 파일 존재하지 않음")

    return "\n".join(results)


# [U-12] 2.8 /etc/services 파일 소유자 및 권한 설정
def check_u12():
    file = "/etc/services"

    if not os.path.exists(file):
        return "U-12,취약,/etc/services 파일이 존재하지 않음"

    stat_info = os.stat(file)
    owner = pwd.getpwuid(stat_info.st_uid).pw_name
    group = grp.getgrgid(stat_info.st_gid).gr_name
    perm = stat.S_IMODE(stat_info.st_mode)
```

```
    if owner == "root" and group == "root" and perm <= 0o644:
        return "U-12,양호,/etc/services 파일 소유자 및 권한 설정 적절"
    else:
        return "U-12,취약,/etc/services 파일 소유자 및 권한 설정 미흡"


# [U-13] 2.9 SUID, SGID 설정 파일 점검
def check_u13():
    suid_files = subprocess.getoutput("find / -type f -perm -4000 2>/dev/nu
ll")
    sgid_files = subprocess.getoutput("find / -type f -perm -2000 2>/dev/nu
ll")

    if not suid_files.strip() and not sgid_files.strip():
        return "U-13,양호,SUID, SGID 설정된 파일이 존재하지 않음"
    else:
        return "U-13,취약,SUID, SGID 설정된 파일 존재"


# [U-14] 2.10 사용자, 시스템 시작파일 및 환경파일 소유자 및 권한 설정
def check_u14():
    users_output = subprocess.getoutput("awk -F: '$3 >= 1000 && $1 != \"no
body\" {print $1}' /etc/passwd")
    users = users_output.strip().splitlines()
    files_to_check = [".profile", ".bash_profile", ".bashrc", ".cshrc", ".kshrc"]
    vulnerable_files = []

    for user in users:
        try:
            home_dir = subprocess.getoutput(f"eval echo ~{user}").strip()
        except Exception:
            continue

        for file in files_to_check:
            target_file = os.path.join(home_dir, file)
            if os.path.isfile(target_file):
                stat_info = os.stat(target_file)
```

```
            owner = pwd.getpwuid(stat_info.st_uid).pw_name
            perm = stat_info.st_mode & 0o777
            if (owner != user and owner != "root") or (perm & 0o22):  # 그룹 또는 기타 쓰기 권한
                vulnerable_files.append(target_file)

    if not vulnerable_files:
        return "U-14,양호,홈 디렉터리 환경파일의 소유자 및 권한 설정 적절"
    else:
        files_str = "\\n".join(vulnerable_files)
        return f"U-14,취약,홈 디렉터리 환경파일의 소유자 및 권한 설정 미흡({files_str})"


# [U-15] 2.11 world writable 파일 점검
def check_u15():
    try:
        result = subprocess.run(
            ["find", "/", "-xdev", "-type", "f", "-perm", "-0002"],
            stdout=subprocess.PIPE,
            stderr=subprocess.DEVNULL,
            text=True,
            check=False
        )
        files = result.stdout.strip().split("\n")
        files = [f for f in files if f.strip()]

        if not files:
            return "U-15,양호,world writable 파일이 존재하지 않음"
        else:
            file_list = "\\n".join(files[:10])  # 너무 많을 경우 대비해 최대 10개만 표시
            return f"U-15,취약,world writable 파일 존재({file_list} 등)"
    except Exception as e:
        return f"U-15,오류,world writable 파일 점검 중 오류 발생: {e}"


# [U-16] 2.12 /dev에 존재하지 않는 device 파일 점검
def check_u16():
```

```
    dev_dir = "/dev"
    found_invalid = []

    try:
        for entry in os.listdir(dev_dir):
            path = os.path.join(dev_dir, entry)

            if not os.path.exists(path):
                continue

            try:
                mode = os.stat(path).st_mode
                is_block = stat.S_ISBLK(mode)
                is_char = stat.S_ISCHR(mode)

                # 블록/문자 디바이스인데 ls -l 목록에 없음
                if (is_block or is_char):
                    result = subprocess.run(
                        ["ls", "-l", dev_dir],
                        stdout=subprocess.PIPE,
                        stderr=subprocess.DEVNULL,
                        text=True
                    )
                    if entry not in result.stdout:
                        found_invalid.append(path)
            except Exception:
                continue

    except Exception as e:
        return f"U-16,오류,/dev 파일 점검 중 오류 발생: {e}"

    if not found_invalid:
        return "U-16,양호,/dev에 존재하지 않는 device 파일이 없음"
    else:
        return f"U-16,취약,/dev에 존재하지 않는 device 파일 존재({', '.join(found_i
nvalid[:10])} 등)"
```

```
# [U-17] 2.13 $HOME/.rhosts, hosts.equiv 사용 금지
def check_u17():
    bad_files = []

    def check_file(filepath, expected_owner):
        if not os.path.isfile(filepath):
            return

        try:
            stat_info = os.stat(filepath)
            perm = stat.S_IMODE(stat_info.st_mode)
            owner = pwd.getpwuid(stat_info.st_uid).pw_name

            if owner != expected_owner and owner != "root":
                bad_files.append(f"{filepath}(소유자 오류)")
                return

            if perm > 0o600:
                bad_files.append(f"{filepath}(권한 오류: {oct(perm)})")
                return

            with open(filepath, "r", encoding="utf-8", errors="ignore") as f:
                if any(line.strip().startswith("+") for line in f):
                    bad_files.append(f"{filepath}(+ 설정 포함)")
        except Exception:
            pass

    # /etc 기준
    check_file("/etc/hosts.equiv", "root")
    check_file("/etc/rhosts", "root")

    # 사용자 홈 디렉터리 내 .rhosts
    for user in pwd.getpwall():
        if user.pw_uid >= 1000 and os.path.isdir(user.pw_dir):
            rhosts_path = os.path.join(user.pw_dir, ".rhosts")
            check_file(rhosts_path, user.pw_name)

    if not bad_files:
```

```
        return "U-17,양호,.rhosts, hosts.equiv, rhosts 파일이 없거나 보안 설정이 적절함"
    else:
        result = "U-17,취약,다음 파일이 보안 기준에 부적합함:\n"
        result += "\n".join(f" - {f}" for f in bad_files)
        return result


# [U-18] 2.14 접속 IP 및 포트 제한
def check_u18():
    issue_found = False

    # iptables 설정 확인
    try:
        result = subprocess.run(["iptables", "-L", "INPUT", "-n"], stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
        if result.returncode == 0:
            for line in result.stdout.splitlines():
                if ("ACCEPT" in line or "DROP" in line) and "0.0.0.0/0" in line:
                    issue_found = True
                    break
    except FileNotFoundError:
        pass  # iptables 없음

    # /etc/hosts.allow 내용 확인
    if os.path.exists("/etc/hosts.allow"):
        with open("/etc/hosts.allow", "r", encoding="utf-8", errors="ignore") as f:
            lines = [line for line in f if not line.strip().startswith("#") and line.strip()]
            if len(lines) == 0:
                issue_found = True

    # /etc/hosts.deny 설정 확인
    if os.path.exists("/etc/hosts.deny"):
        with open("/etc/hosts.deny", "r", encoding="utf-8", errors="ignore") as f:
            if not any("ALL: ALL" in line for line in f):
```

```
            issue_found = True

    if issue_found:
        return "U-18,취약,IP 또는 포트 제한 설정이 부적절함"
    else:
        return "U-18,양호,IP 및 포트 접근이 제한되어 있음"


# [U-19] 3.1 Finger 서비스 비활성화
def check_u19():
    try:
        result = subprocess.run(["systemctl", "is-enabled", "finger.service"],
                                stdout=subprocess.PIPE,
                                stderr=subprocess.PIPE,
                                text=True)
        if result.returncode == 0 and result.stdout.strip() == "enabled":
            return "U-19,취약,Finger 서비스가 활성화 되어 있음"
        else:
            return "U-19,양호,Finger 서비스가 비활성화 되어 있음"
    except FileNotFoundError:
        return "U-19,양호,Finger 서비스가 존재하지 않음"


# [U-20] 3.2 Anonymous FTP 비활성화
def check_u20():
    import subprocess
    import os

    try:
        result = subprocess.run(["systemctl", "is-active", "vsftpd.service"],
                                stdout=subprocess.PIPE,
                                stderr=subprocess.DEVNULL,
                                text=True)
        if result.returncode == 0 and result.stdout.strip() == "active":
            try:
                with open("/etc/vsftpd/vsftpd.conf", "r") as f:
                    content = f.read()
                    if "anonymous_enable=NO" in content:
```

```
                return "U-20,양호,익명 FTP 접속이 차단되어 있음"
            else:
                return "U-20,취약,익명 FTP 접속이 허용되어 있음"
        except FileNotFoundError:
            return "U-20,취약,vsftpd 설정 파일이 존재하지 않음"
    else:
        return "U-20,양호,vsftpd 서비스 비활성화"
except FileNotFoundError:
    return "U-20,양호,systemctl 또는 vsftpd가 설치되어 있지 않음"


# [U-21] 3.3 r 계열 서비스 비활성화
def check_u21():
    import subprocess

    services = ["rlogin.service", "rsh.service", "rexec.service"]
    for svc in services:
        result = subprocess.run(["systemctl", "is-enabled", svc],
                                stdout=subprocess.DEVNULL,
                                stderr=subprocess.DEVNULL)
        if result.returncode == 0:
            return "U-21,취약,불필요한 r 계열 서비스가 활성화 되어 있음"

    return "U-21,양호,불필요한 r 계열 서비스가 비활성화 되어 있음"


# [U-22] 3.4 crond 파일 소유자 및 권한 설정
def check_u22():
    import shutil
    import os
    import stat

    issue_found = False

    crontab_path = shutil.which("crontab")
    if crontab_path:
        crontab_perm = os.stat(crontab_path).st_mode & 0o777
        if crontab_perm > 0o750:
```

```
            issue_found = True
    else:
        issue_found = True

    cron_files = [
        "/etc/crontab",
        "/etc/cron.allow",
        "/etc/cron.d",
        "/var/spool/cron"
    ]

    for f in cron_files:
        if os.path.exists(f):
            perm = os.stat(f).st_mode & 0o777
            if perm > 0o640:
                issue_found = True
                break

    if not issue_found:
        return "U-22,양호,crontab 명령어 일반사용자 금지 및 cron 관련 파일 권한 적
정"
    else:
        return "U-22,취약,crontab 권한 또는 cron 파일 권한 부적절"


# [U-23] 3.5 DoS 공격에 취약한 서비스 비활성화
def check_u23():
    import subprocess

    services = [
        "chargen-dgram@udp.service",
        "chargen-stream@tcp.service",
        "daytime-dgram@udp.service",
        "daytime-stream@tcp.service",
        "echo-dgram@udp.service",
        "echo-stream@tcp.service",
        "tcpmux-server.service"
    ]
```

```
    for svc in services:
        try:
            result = subprocess.run(["systemctl", "is-enabled", svc],
                                    stdout=subprocess.DEVNULL,
                                    stderr=subprocess.DEVNULL)
            if result.returncode == 0:
                return "U-23,취약,DoS 공격에 취약한 서비스가 활성화 되어 있음"
        except Exception:
            continue

    return "U-23,양호,DoS 공격에 취약한 서비스가 비활성화 되어 있음"


# [U-24] 3.6 NFS 서비스 비활성화
def check_u24():
    import subprocess

    services = [
        "nfs-server.service",
        "rpcbind.service"
    ]

    for svc in services:
        try:
            result = subprocess.run(
                ["systemctl", "is-enabled", svc],
                stdout=subprocess.DEVNULL,
                stderr=subprocess.DEVNULL
            )
            if result.returncode == 0:
                return "U-24,취약,불필요한 NFS 서비스 관련 데몬이 활성화 되어 있음"
        except Exception:
            continue

    return "U-24,양호,불필요한 NFS 서비스 관련 데몬이 비활성화 되어 있음"
```

```
# [U-25] 3.7 NFS 접근 통제
def check_u25():
    exports_file = "/etc/exports"

    try:
        with open(exports_file, "r") as f:
            lines = f.readlines()

        for line in lines:
            line = line.strip()
            if line.startswith("#") or not line:
                continue
            if "*" in line:
                return "U-25,취약,/etc/exports 파일에 everyone(*) 공유 설정이 있음"
        return "U-25,양호,/etc/exports 파일에 everyone(*) 공유 설정이 없음"

    except FileNotFoundError:
        return "U-25,양호,/etc/exports 파일이 존재하지 않음 (NFS 사용 안 함)"


# [U-26] 3.8 automountd 제거
def check_u26():
    try:
        # 설치 여부 확인
        result = subprocess.run(
            ["systemctl", "list-unit-files"],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True
        )

        if "autofs" not in result.stdout:
            return "U-26,양호,autofs(automountd) 서비스가 설치되어 있지 않음"

        # 서비스 활성화 여부 확인
        state_result = subprocess.run(
            ["systemctl", "is-enabled", "autofs"],
            stdout=subprocess.PIPE,
```

```
            stderr=subprocess.PIPE,
            text=True
        )
        state = state_result.stdout.strip()

        if state == "enabled":
            return "U-26,취약,autofs(automountd) 서비스가 활성화되어 있음"
        else:
            return "U-26,양호,autofs(automountd) 서비스가 비활성화되어 있음"

    except Exception:
        return "U-26,양호,autofs(automountd) 서비스 상태 확인 중 오류 발생"


# [U-27] 3.9 RPC 서비스 확인
def check_u27():
    services = ["rpcbind", "rpc-statd", "rpc-idmapd"]
    vulnerable = []

    try:
        listed_units = subprocess.run(
            ["systemctl", "list-unit-files"],
            stdout=subprocess.PIPE,
            stderr=subprocess.DEVNULL,
            text=True
        ).stdout

        for svc in services:
            if svc in listed_units:
                state_result = subprocess.run(
                    ["systemctl", "is-enabled", svc],
                    stdout=subprocess.PIPE,
                    stderr=subprocess.DEVNULL,
                    text=True
                )
                state = state_result.stdout.strip()
                if state == "enabled":
                    vulnerable.append(svc)
```

```
        if not vulnerable:
            return "U-27,양호,불필요한 RPC 서비스가 비활성화되어 있음"
        else:
            return f"U-27,취약,다음 RPC 서비스가 활성화되어 있음 ({', '.join(vulnerable)})"

    except Exception:
        return "U-27,취약,RPC 서비스 확인 중 오류 발생"


# [U-28] 3.10 NIS, NIS+ 점검
def check_u28():
    services = ["ypserv", "ypbind", "yppasswdd", "rpc.yppasswdd"]
    vulnerable = []

    try:
        listed_units = subprocess.run(
            ["systemctl", "list-unit-files"],
            stdout=subprocess.PIPE,
            stderr=subprocess.DEVNULL,
            text=True
        ).stdout

        for svc in services:
            if svc in listed_units:
                state_result = subprocess.run(
                    ["systemctl", "is-enabled", svc],
                    stdout=subprocess.PIPE,
                    stderr=subprocess.DEVNULL,
                    text=True
                )
                if state_result.stdout.strip() == "enabled":
                    vulnerable.append(svc)

        if not vulnerable:
            return "U-28,양호,NIS 서비스가 비활성화되어 있음"
        else:
```

```
            return f"U-28,취약,다음 NIS 서비스가 활성화되어 있음 ({', '.join(vulnerabl
e)})"

    except Exception:
        return "U-28,취약,NIS 서비스 점검 중 오류 발생"


#[U-29] 3.11 tftp, talk 서비스 비활성화
def check_u29():
    services = ["tftp", "talk", "ntalk"]
    vulnerable = []

    try:
        listed_units = subprocess.run(
            ["systemctl", "list-unit-files"],
            stdout=subprocess.PIPE,
            stderr=subprocess.DEVNULL,
            text=True
        ).stdout

        for svc in services:
            if svc in listed_units:
                result = subprocess.run(
                    ["systemctl", "is-enabled", svc],
                    stdout=subprocess.PIPE,
                    stderr=subprocess.DEVNULL,
                    text=True
                )
                if result.stdout.strip() == "enabled":
                    vulnerable.append(svc)

        if not vulnerable:
            return "U-29,양호,tftp, talk, ntalk 서비스가 비활성화되어 있음"
        else:
            return f"U-29,취약,다음 서비스가 활성화되어 있음 ({', '.join(vulnerabl
e)})"

    except Exception:
```

```
        return "U-29,취약,서비스 점검 중 오류 발생"


# [U-30] Sendmail 버전 점검
def check_u30():
    recommended_version = "8.18.1"

    # sendmail 명령 존재 여부 확인
    if shutil.which("sendmail") is None:
        return "U-30,양호,Sendmail이 설치되어 있지 않음"

    try:
        result = subprocess.run(
            ["sendmail", "-d0.1", "-bv", "root"],
            stdout=subprocess.PIPE,
            stderr=subprocess.DEVNULL,
            text=True
        )
        for line in result.stdout.splitlines():
            if "Version" in line:
                current_version = line.split()[1]
                if current_version == recommended_version:
                    return f"U-30,양호,Sendmail이 최신 버전({recommended_versio
n})"
                else:
                    return f"U-30,취약,Sendmail 버전이 최신이 아님 (현재: {current_ve
rsion}, 권장: {recommended_version})"
        return "U-30,취약,Sendmail 버전 정보를 확인할 수 없음"
    except Exception:
        return "U-30,취약,Sendmail 버전 점검 중 오류 발생"


# [U-31] 3.13 스팸 메일 릴레이 제한
def is_enabled(service):
    try:
        result = subprocess.run(["systemctl", "is-enabled", service], stdout=s
ubprocess.PIPE, stderr=subprocess.DEVNULL, text=True)
        return result.stdout.strip() == "enabled"
```

```
    except Exception:
        return False

def check_sendmail():
    try:
        result = subprocess.run(["grep", "-i", "PrivacyOptions", "/etc/mail/sen
dmail.mc"], stdout=subprocess.PIPE, stderr=subprocess.DEVNULL, text=Tr
ue)
        return "noexpn" in result.stdout and "restrictmailq" in result.stdout
    except Exception:
        return False

def check_postfix():
    try:
        result = subprocess.run(["grep", "-i", "smtpd_recipient_restrictions", "/
etc/postfix/main.cf"], stdout=subprocess.PIPE, stderr=subprocess.DEVNU
LL, text=True)
        return "reject_unauth_destination" in result.stdout.lower()
    except Exception:
        return False

def check_u31():
    vulnerable = []

    for svc in ["sendmail", "postfix"]:
        if shutil.which(svc):
            if is_enabled(svc):
                if svc == "sendmail" and not check_sendmail():
                    vulnerable.append(svc)
                elif svc == "postfix" and not check_postfix():
                    vulnerable.append(svc)

    if not vulnerable:
        return "U-31,양호,SMTP 서비스가 없거나 릴레이 제한이 설정되어 있음"
    else:
        return f"U-31,취약,다음 SMTP 서비스에서 릴레이 제한이 설정되어 있지 않음
({', '.join(vulnerable)})"
```

```
# [U-32] 3.14 일반사용자의 Sendmail 실행 방지
import subprocess
import os

def check_u32():
    service = "sendmail"
    sendmail_cf = "/etc/mail/sendmail.cf"

    try:
        result = subprocess.run(["systemctl", "list-unit-files"], stdout=subproc
ess.PIPE, stderr=subprocess.DEVNULL, text=True)
        if service not in result.stdout:
            return "U-32,양호,Sendmail이 설치되어 있지 않음 또는 사용하지 않음"
    except Exception:
        return "U-32,양호,Sendmail 상태 확인 중 오류 또는 비설치"

    if os.path.exists(sendmail_cf):
        try:
            with open(sendmail_cf, 'r') as f:
                for line in f:
                    if line.strip().startswith("O RestrictMailq=restrictqrun"):
                        return "U-32,양호,일반 사용자의 Sendmail 실행이 제한되어 있음"
        except Exception:
            pass

    return "U-32,취약,일반 사용자의 Sendmail 실행 제한이 설정되어 있지 않음"


# [U-33] 3.15 DNS 보안 버전 패치
def check_u33():
    service = "named"
    recommended_version = "9.18.24"

    if shutil.which(service) is None:
        return "U-33,양호,DNS 서비스를 사용하지 않음"

    try:
```

```
        result = subprocess.run([service, "-v"], stdout=subprocess.PIPE, stde
rr=subprocess.DEVNULL, text=True)
        version = result.stdout.strip().split()[1]
    except Exception:
        return "U-33,취약,DNS 버전 확인 실패"

    if version == recommended_version:
        return f"U-33,양호,DNS 서비스가 최신 버전({recommended_version})"
    else:
        return f"U-33,취약,DNS 서비스 버전이 최신이 아님 (현재: {version}, 권장: {r
ecommended_version})"


# [U-34] 3.16 DNS Zone Transfer 설정
def check_u34():
    config_file = "/etc/named.conf"

    if shutil.which("named") is None:
        return "U-34,양호,DNS 서비스를 사용하지 않음"

    if not os.path.isfile(config_file):
        return "U-34,취약,named.conf 파일이 존재하지 않음"

    try:
        with open(config_file, 'r') as f:
            lines = f.readlines()
    except Exception:
        return "U-34,취약,named.conf 파일을 읽는 중 오류 발생"

    allow_transfer_lines = [line for line in lines if "allow-transfer" in line and n
ot line.strip().startswith("#")]

    if not allow_transfer_lines:
        return "U-34,취약,Zone Transfer 설정이 없음 (기본값은 모든 사용자에게 허
용)"

    if any("any" in line for line in allow_transfer_lines):
        return "U-34,취약,Zone Transfer가 모든 사용자(any)에게 허용됨"
```

```
    else:
        return "U-34,양호,Zone Transfer가 허가된 사용자에게만 허용됨"


# [U-35] 3.17 웹서비스 디렉토리 리스팅 제거
def check_u35():
    httpd_conf = "/etc/conf/httpd.conf"

    if shutil.which("httpd") is None:
        return "U-35,양호,웹서비스(Apache)를 사용하지 않음"

    if not os.path.isfile(httpd_conf):
        return "U-35,취약,Apache 설정 파일이 존재하지 않음"

    try:
        with open(httpd_conf, "r") as f:
            for line in f:
                if "Options Indexes" in line and not line.strip().startswith("#"):
                    return "U-35,취약,디렉토리 리스팅(Indexes)이 허용되어 있음"
    except Exception:
        return "U-35,취약,httpd.conf 파일 읽기 실패"

    return "U-35,양호,디렉토리 리스팅이 제거되어 있음"


# [U-36] 3.18 웹서비스 웹 프로세스 권한 제한
def check_u36():
    web_processes = ["httpd", "apache2"]

    try:
        # 프로세스 확인
        proc_list = subprocess.run(["ps", "-eo", "user:20,comm"], stdout=sub
process.PIPE, text=True)
        lines = proc_list.stdout.strip().split("\n")

        # 웹 프로세스 실행 사용자 추출
        running_users = set()
        for line in lines:
```

```
            for proc in web_processes:
                if proc in line and "root" not in line:
                    running_users.add(line.split()[0])

        # 프로세스 자체가 안 돌고 있으면
        httpd_running = subprocess.run(["pgrep", "-x", "httpd"], stdout=subpr
ocess.DEVNULL).returncode == 0
        apache_running = subprocess.run(["pgrep", "-x", "apache2"], stdout=
subprocess.DEVNULL).returncode == 0

        if not httpd_running and not apache_running:
            return "U-36,양호,웹서비스가 구동되고 있지 않음"

        if "root" in running_users:
            return "U-36,취약,웹 프로세스가 root 권한으로 실행 중"

        if running_users:
            return f"U-36,양호,웹 프로세스가 일반 사용자({', '.join(running_users)})
권한으로 실행 중"

        return "U-36,취약,웹 프로세스 실행 사용자 확인 불가"

    except Exception:
        return "U-36,취약,웹 프로세스 상태 확인 중 오류 발생"


# [U-37] 3.19 웹서비스 상위 디렉토리 접근 금지
def check_u37():
    httpd_conf = "/etc/httpd/conf/httpd.conf"
    access_conf = "/etc/httpd/conf.d/access.conf"
    file_to_check = ""

    if os.path.isfile(httpd_conf):
        file_to_check = httpd_conf
    elif os.path.isfile(access_conf):
        file_to_check = access_conf

    if not file_to_check:
```

```
        return "U-37,양호,Apache 설정 파일이 존재하지 않음 (웹 서비스 미사용)"

    try:
        with open(file_to_check, 'r') as f:
            content = f.read()
            if re.search(r'<Directory\s+.*\.\..*>', content, re.IGNORECASE):
                return "U-37,취약,상위 디렉토리 접근 허용 설정이 존재함"
    except Exception:
        return "U-37,취약,설정 파일 확인 중 오류 발생"

    return "U-37,양호,상위 디렉토리 접근 제한 설정이 적용됨"


# [U-38] 3.20 웹서비스 불필요한 파일 제거
def check_u38():
    apache_paths = [
        "/var/www/html/manual",
        "/var/www/manual",
        "/etc/httpd/htdocs/manual",
        "/etc/httpd/manual"
    ]
    vulnerable = []

    for path in apache_paths:
        if os.path.exists(path):
            vulnerable.append(path)

    if not vulnerable:
        return "U-38,양호,불필요한 Apache 매뉴얼 디렉터리가 존재하지 않음"
    else:
        return f"U-38,취약,다음 불필요한 디렉터리가 존재함 ({', '.join(vulnerabl
e)})"


# [U-39] 3.21 웹서비스 링크 사용금지
def check_u39():
    apache_conf = "/etc/httpd/conf/httpd.conf"
    vulnerable = []
```

```
    if not os.path.isfile(apache_conf):
        return "U-39,양호,Apache 설정 파일이 존재하지 않음 (웹 서비스 미사용)"

    try:
        with open(apache_conf, "r") as f:
            for line in f:
                if re.search(r'Options\s+.*FollowSymLinks', line):
                    vulnerable.append("FollowSymLinks 옵션")
                if re.search(r'^\s*Alias\s+', line):
                    vulnerable.append("Alias 설정")
    except Exception:
        return "U-39,취약,Apache 설정 파일을 읽을 수 없음"

    if not vulnerable:
        return "U-39,양호,심볼릭 링크(FollowSymLinks) 및 Alias 사용이 제한됨"
    else:
        return f"U-39,취약,다음 항목이 설정되어 있음 ({', '.join(vulnerable)})"


# [U-40] 3.22 웹서비스 파일 업로드 및 다운로드 제한
def check_u40():
    apache_conf = "/etc/httpd/conf/httpd.conf"

    if not os.path.isfile(apache_conf):
        return "U-40,양호,Apache 설정 파일이 존재하지 않음 (웹 서비스 미사용)"

    limit_value = None
    with open(apache_conf, 'r') as f:
        for line in f:
            match = re.match(r'^\s*LimitRequestBody\s+(\d+)', line)
            if match:
                limit_value = int(match.group(1))
                break

    if limit_value is None:
        return "U-40,취약,LimitRequestBody 설정이 없음"
    elif limit_value <= 5000000:
```

```
        return f"U-40,양호,LimitRequestBody가 {limit_value} 바이트로 제한됨"
    else:
        return f"U-40,취약,LimitRequestBody가 {limit_value} 바이트로 너무 크게 설정됨"


# [U-41] 3.23 웹서비스 영역의 분리
def check_u41():
    apache_conf = "/etc/httpd/conf/httpd.conf"
    default_paths = [
        "/usr/local/apache/htdocs",
        "/usr/local/apache2/htdocs",
        "/var/www/html"
    ]

    if not os.path.isfile(apache_conf):
        return "U-41,양호,Apache 설정 파일이 존재하지 않음 (웹 서비스 미사용)"

    docroot = None
    with open(apache_conf, 'r') as f:
        for line in f:
            match = re.match(r'^\s*DocumentRoot\s+"([^"]+)"', line)
            if match:
                docroot = match.group(1)
                break

    if not docroot:
        return "U-41,취약,DocumentRoot 설정이 없음"
    elif docroot in default_paths:
        return f"U-41,양호,DocumentRoot가 별도 디렉터리로 지정됨 ({docroot})"
    else:
        return f"U-41,취약,DocumentRoot가 기본 경로가 아님 ({docroot})"


# [U-42] 4.1 최신 보안패치 및 벤더 권고사항 적용
def check_u42():
    policy_file = "/etc/security/patch_policy.conf"
    patch_log = "/var/log/patch_management.log"
```

```
    policy_status = "수립됨" if os.path.isfile(policy_file) else "수립되지 않음"
    patch_status = "주기적으로 패치 적용 확인됨" if os.path.isfile(patch_log) an
d os.path.getsize(patch_log) > 0 else "패치 적용 내역 확인 불가 또는 없음"

    if policy_status == "수립됨" and patch_status == "주기적으로 패치 적용 확인
됨":
        return "U-42,양호,패치 정책이 수립되어 있으며, 정책에 따른 패치 적용 내역이
존재함"
    else:
        return f"U-42,취약,패치 정책 수립 또는 적용 내역이 부적절함 (정책: {policy_s
tatus}, 적용: {patch_status})"


# [U-43] 5.1 로그의 정기적 검토 및 보고
def check_u43():
    vulnerable = []
    logs = {
        "/var/log/wtmp": "wtmp 로그 없음",
        "/var/log/btmp": "btmp 로그 없음",
        "/var/log/sulog": "sulog 로그 없음",
        "/var/log/xferlog": "xferlog 로그 없음"
    }

    for path, message in logs.items():
        try:
            with open(path, "rb") as f:
                f.read(1)
        except:
            vulnerable.append(message)

    if not vulnerable:
        return "U-43,양호,로그 파일들이 정상적으로 존재하며 검토 가능"
    else:
        return f"U-43,취약,누락된 로그 파일 - {', '.join(vulnerable)}"


# [U-44] 1.5 root 이외의 UID가 ‘0’ 금지
```

```
def check_u44():
    vulnerable = []
    with open("/etc/passwd", "r") as f:
        for line in f:
            parts = line.strip().split(":")
            if len(parts) > 2:
                username, uid = parts[0], parts[2]
                if uid == "0" and username != "root":
                    vulnerable.append(username)

    if not vulnerable:
        return "U-44,양호,root 외에 UID 0을 가진 계정이 없음"
    else:
        return f"U-44,취약,root 외에 UID 0을 가진 계정 존재 ({', '.join(vulnerabl
e)})"


# [U-45] 1.6 root 계정 su 제한
def check_u45():
    pam_file = "/etc/pam.d/su"
    group_name = "wheel"

    try:
        with open(pam_file, "r") as f:
            for line in f:
                if line.strip().startswith("auth") and "pam_wheel.so" in line:
                    return f"U-45,양호,su 명령어 사용이 '{group_name}' 그룹으로 제한
되어 있음"
        return "U-45,취약,su 명령어 사용 제한 설정이 없음"
    except FileNotFoundError:
        return f"U-45,취약,{pam_file} 파일이 존재하지 않음"


# [U-46] 1.7 패스워드 최소 길이 설정
def check_u46():
    login_defs = "/etc/login.defs"
    minlen = None
```

```
    try:
        with open(login_defs, 'r') as f:
            for line in f:
                if line.strip().startswith("PASS_MIN_LEN"):
                    parts = line.strip().split()
                    if len(parts) >= 2:
                        minlen = parts[1]
                    break
    except FileNotFoundError:
        return f"U-46,취약,{login_defs} 파일이 존재하지 않음"

    if minlen is None:
        return "U-46,취약,패스워드 최소 길이 설정이 존재하지 않음"

    try:
        if int(minlen) >= 8:
            return "U-46,양호,패스워드 최소 길이가 8자 이상으로 설정되어 있음"
        else:
            return f"U-46,취약,패스워드 최소 길이가 8자 미만으로 설정되어 있음 (현재:
{minlen})"
    except ValueError:
        return f"U-46,취약,패스워드 최소 길이 값이 올바르지 않음 (현재: {minlen})"


# [U-47] 1.8 패스워드 최대 사용기간 설정
def check_u47():
    login_defs = "/etc/login.defs"
    maxdays = None

    try:
        with open(login_defs, "r") as f:
            for line in f:
                if line.strip().startswith("PASS_MAX_DAYS"):
                    parts = line.strip().split()
                    if len(parts) >= 2:
                        maxdays = parts[1]
                    break
    except FileNotFoundError:
```

```
        return f"U-47,취약,{login_defs} 파일이 존재하지 않음"

    if maxdays is None:
        return "U-47,취약,패스워드 최대 사용기간 설정이 존재하지 않음"

    try:
        if int(maxdays) <= 90:
            return f"U-47,양호,패스워드 최대 사용기간이 90일 이하로 설정되어 있음
(현재: {maxdays})"
        else:
            return f"U-47,취약,패스워드 최대 사용기간이 90일 초과로 설정되어 있음
(현재: {maxdays})"
    except ValueError:
        return f"U-47,취약,패스워드 최대 사용기간 값이 올바르지 않음 (현재: {maxda
ys})"


# [U-48] 1.9 패스워드 최소 사용기간 설정
def check_u48():
    vulnerable_users = []

    try:
        with open("/etc/passwd", "r") as f:
            for line in f:
                parts = line.strip().split(":")
                if len(parts) >= 3:
                    username = parts[0]
                    uid = int(parts[2])
                    if uid >= 1000 and username != "nobody":
                        # chage 명령으로 최소 사용일 확인
                        try:
                            from subprocess import run, PIPE
                            result = run(["chage", "-l", username], stdout=PIPE, stderr
=PIPE, text=True)
                            for ch_line in result.stdout.splitlines():
                                if "Minimum number of days between password chang
e" in ch_line:
                                    value = ch_line.split(":")[1].strip()
```

```
                        if value == "":
                            vulnerable_users.append(f"{username} (설정 확인 불
가)")
                        elif int(value) < 1:
                            vulnerable_users.append(f"{username} ({value}
일)")
                        break
                except Exception:
                    vulnerable_users.append(f"{username} (chage 오류)")
    except Exception:
        return "U-48,취약,/etc/passwd 파일을 읽을 수 없음"

    if not vulnerable_users:
        return "U-48,양호,모든 일반 사용자의 패스워드 최소 사용기간이 1일 이상으로
설정됨"
    else:
        msg = "U-48,취약,다음 사용자들의 최소 사용기간이 기준 미만임\n"
        msg += "\n".join([f" - {u}" for u in vulnerable_users])
        return msg


#[U-49] 1.10 불필요한 계정 제거
def check_u49():
    default_accounts = {"lp", "uucp", "nuucp", "sync", "shutdown", "halt", "n
ews", "operator", "games", "gopher"}
    suspicious_accounts = []
    unused_accounts = []

    # /etc/passwd의 모든 계정 이름 가져오기
    try:
        with open("/etc/passwd", "r") as f:
            users = [line.split(":")[0] for line in f]

        for user in users:
            # default account 체크
            if user in default_accounts:
                suspicious_accounts.append(user)
```

```
            # 로그인 기록이 없는 계정 확인
            try:
                result = subprocess.run(["lastlog", "-u", user], capture_output=Tr
ue, text=True)
                if "Never logged in" in result.stdout:
                    unused_accounts.append(user)
            except Exception as e:
                continue  # lastlog 명령 실패 시 해당 사용자 무시

        if not suspicious_accounts and not unused_accounts:
            return "U-49,양호,불필요한 계정이 존재하지 않음"
        else:
            return f"U-49,취약,불필요한 계정 또는 미사용 계정 존재 (기준계정: {', '.joi
n(suspicious_accounts)}, 미사용계정: {', '.join(unused_accounts)})"

    except Exception as e:
        return f"U-49,오류,점검 중 예외 발생: {e}"


# [U-50] 1.11 관리자 그룹에 최소한의 계정 포함
def check_u50():
    admin_group = "root"
    group_file = "/etc/group"

    try:
        with open(group_file, "r") as f:
            for line in f:
                if line.startswith(f"{admin_group}:"):
                    parts = line.strip().split(":")
                    if len(parts) >= 4:
                        accounts = parts[3]
                        if accounts.strip() == "":
                            return f"U-50,취약,관리자 그룹 '{admin_group}'에 등록된 계정
이 없음"
                        else:
                            return f"U-50,양호,관리자 그룹 '{admin_group}'에 등록된 계정
이 있음 ({accounts.strip()})"
                    else:
```

```
                    return f"U-50,취약,관리자 그룹 '{admin_group}'의 항목 형식이 비정상적임"
        return f"U-50,취약,관리자 그룹 '{admin_group}'이 존재하지 않음"
    except Exception:
        return "U-50,취약,관리자 그룹 정보 확인 중 오류 발생"


# [U-51] 1.12 계정이 존재하지 않는 GID 금지
def check_u51():
    group_file = "/etc/group"
    passwd_file = "/etc/passwd"
    invalid_gids = []

    try:
        # 그룹 파일에서 GID 목록 추출
        with open(group_file, "r") as gf:
            group_gids = set()
            for line in gf:
                parts = line.strip().split(":")
                if len(parts) >= 3:
                    group_gids.add(parts[2])

        # 패스워드 파일에서 사용 중인 GID 추출
        with open(passwd_file, "r") as pf:
            used_gids = set()
            for line in pf:
                parts = line.strip().split(":")
                if len(parts) >= 4:
                    used_gids.add(parts[3])

        # 계정이 참조하지 않는 GID 판별
        for gid in group_gids:
            if gid not in used_gids:
                invalid_gids.append(gid)

        if not invalid_gids:
            return "U-51,양호,계정이 존재하지 않는 GID가 없음"
        else:
```

```
            return f"U-51,취약,계정이 존재하지 않는 GID가 존재함 ({', '.join(invalid_gi
ds)})"

    except Exception as e:
        return f"U-51,오류,점검 중 예외 발생: {e}"


# [U-52] 1.13 동일한 UID 금지
def check_u52():
    passwd_file = "/etc/passwd"
    uid_map = {}
    duplicates = []

    try:
        with open(passwd_file, "r") as f:
            for line in f:
                parts = line.strip().split(":")
                if len(parts) < 3:
                    continue
                username, uid = parts[0], parts[2]
                if uid in uid_map:
                    duplicates.append(f"{uid} ({uid_map[uid]}, {username})")
                else:
                    uid_map[uid] = username

        if not duplicates:
            return "U-52,양호,동일한 UID가 존재하지 않습니다."
        else:
            result = "U-52,취약,동일한 UID가 중복된 계정이 있습니다. 중복 UID 목록:"
            for entry in duplicates:
                result += f"\n - {entry}"
            return result
    except Exception as e:
        return f"U-52,오류,점검 중 예외 발생: {e}"


# [U-53] 1.14 사용자 shell 점검
def check_u53():
```

```
    nologin_users = ["bin", "daemon", "adm", "lp", "sync", "shutdown", "halt",
"mail", "operator", "games", "ftp", "nobody"]
    misconfigured = []

    try:
        with open("/etc/passwd", "r") as f:
            passwd_lines = f.readlines()

        for line in passwd_lines:
            parts = line.strip().split(":")
            if len(parts) < 7:
                continue
            username, shell = parts[0], parts[-1]
            if username in nologin_users and shell not in ["/sbin/nologin", "/bin/f
alse"]:
                misconfigured.append(f"{username} ({shell})")

        if not misconfigured:
            return "U-53,양호,로그인 불필요 계정에 적절한 쉘이 설정되어 있습니다."
        else:
            result = "U-53,취약,다음 계정에 적절하지 않은 쉘이 설정되어 있습니다:"
            for entry in misconfigured:
                result += f"\n - {entry}"
            return result

    except Exception as e:
        return f"U-53,오류,점검 중 예외 발생: {e}"


# [U-54] 1.15 Session Timeout 설정
def check_u54():
    files = ["/etc/profile", "/etc/bashrc"] + glob.glob("/etc/profile.d/*.sh")
    is_secure = True
    details = []

    for file in files:
        if not os.path.isfile(file):
            continue
```

```
        try:
            with open(file, "r") as f:
                lines = f.readlines()

            tmout_line = ""
            for line in lines:
                if "TMOUT" in line and "=" in line and not line.strip().startswith
("#"):
                    tmout_line = line.strip()
                    break

            if tmout_line:
                try:
                    value = int(tmout_line.split("=")[1].strip())
                    if value > 600:
                        is_secure = False
                        details.append(f"{file}: TMOUT={value} (600초 초과)")
                except ValueError:
                    is_secure = False
                    details.append(f"{file}: TMOUT 값이 숫자가 아님")
            else:
                is_secure = False
                details.append(f"{file}: TMOUT 설정 없음")
        except Exception as e:
            is_secure = False
            details.append(f"{file}: 파일 처리 중 오류 발생 ({e})")

    if is_secure:
        return "U-54,양호,Session Timeout이 적절하게 설정되어 있습니다."
    else:
        result = "U-54,취약,다음 파일에 설정이 누락되었거나 600초 초과 설정이 존재
합니다."
        for d in details:
            result += f"\n - {d}"
        return result


# [U-55] 2.15 hosts.lpd 파일 소유자 및 권한 설정
```

```
def check_u55():
    file = "/etc/hosts.lpd"

    if not os.path.exists(file):
        return "U-55,양호,hosts.lpd 파일이 존재하지 않습니다."

    try:
        st = os.stat(file)
        owner = pwd.getpwuid(st.st_uid).pw_name
        perm = oct(st.st_mode & 0o777)[2:]

        if owner == "root" and perm == "600":
            return "U-55,양호,hosts.lpd 파일의 소유자 및 권한이 적절합니다."
        else:
            return (
                "U-55,취약,hosts.lpd 파일의 보안 설정이 적절하지 않습니다.\n"
                f" - 현재 소유자: {owner}\n"
                f" - 현재 권한: {perm}"
            )
    except Exception as e:
        return f"U-55,오류,hosts.lpd 파일 점검 중 예외 발생: {e}"


# [U-56] 2.17 UMASK 설정 관리
def check_u56():
    files = [
        "/etc/profile",
        "/etc/bashrc",
        "/etc/login.defs",
        "/etc/csh.cshrc"
    ]
    vulnerable = []
    found_valid = False

    for file in files:
        if os.path.isfile(file):
            try:
                with open(file, 'r') as f:
```

```
                for line in f:
                    if re.match(r'^\s*umask\s+[0-7]{3}', line) and not line.strip().
startswith('#'):
                        umask_val = re.search(r'([0-7]{3})', line)
                        if umask_val:
                            val = int(umask_val.group(1), 8)
                            if val < 0o22:
                                vulnerable.append(f"{file}: {line.strip()}")
                            else:
                                found_valid = True
            except Exception:
                continue

    if vulnerable:
        result = "U-56,취약,다음 파일에서 umask 설정이 022 미만입니다:\n"
        result += '\n'.join(f" - {v}" for v in vulnerable)
        return result
    elif found_valid:
        return "U-56,양호,모든 umask 설정이 022 이상입니다."
    else:
        return "U-56,취약,명시적인 UMASK 설정이 없습니다."


# [U-57] 2.18 홈디렉토리 소유자 및 권한 설정
def check_u57():
    vulnerable = []

    with open("/etc/passwd", "r") as f:
        for line in f:
            parts = line.strip().split(":")
            if len(parts) < 6:
                continue
            username, _, uid, _, _, homedir, _ = parts
            try:
                if int(uid) >= 1000 and os.path.isdir(homedir):
                    stat_info = os.stat(homedir)
                    owner = pwd.getpwuid(stat_info.st_uid).pw_name
                    perm = oct(stat.S_IMODE(stat_info.st_mode))[-3:]
```

```
                other_write = int(perm[-1])

                if owner != username or other_write >= 2:
                    vulnerable.append(f"{username} ({homedir} - owner: {owne
r}, perm: {perm})")
            except Exception:
                continue

    if not vulnerable:
        return "U-57,양호,모든 홈 디렉토리 소유자와 권한이 적절합니다."
    else:
        result = "U-57,취약,다음 계정의 홈 디렉토리 설정이 잘못되어 있습니다:\n"
        result += "\n".join(f" - {v}" for v in vulnerable)
        return result


# [U-58] 2.19 홈디렉토리로 지정한 디렉토리의 존재 관리
def check_u58():
    vuln_users = []

    try:
        with open("/etc/passwd", "r") as f:
            for line in f:
                parts = line.strip().split(":")
                if len(parts) < 7:
                    continue
                username, _, uid, _, _, home, shell = parts
                if username == "nobody":
                    continue
                try:
                    if int(uid) >= 1000 and not os.path.isdir(home):
                        vuln_users.append(f"{username}({home})")
                except ValueError:
                    continue
    except Exception:
        return "U-58,취약,/etc/passwd 파일을 읽는 중 오류 발생"

    if not vuln_users:
```

```
        return "U-58,양호,모든 계정의 홈 디렉터리가 존재합니다."
    else:
        result = "U-58,취약,다음 계정의 홈 디렉터리가 존재하지 않습니다:\n"
        result += "\n".join(f" - {user}" for user in vuln_users)
        return result


# [U-59] 2.20 숨겨진 파일 및 디렉토리 검색 및 제거
def check_u59():
    import os

    hidden_files = []
    hidden_dirs = []

    for root, dirs, files in os.walk("/", topdown=True):
        # 숨겨진 디렉토리 필터링
        for d in dirs:
            if d.startswith(".") and d not in (".", ".."):
                hidden_dirs.append(os.path.join(root, d))
        # 숨겨진 파일 필터링
        for f in files:
            if f.startswith("."):
                hidden_files.append(os.path.join(root, f))

    if not hidden_files and not hidden_dirs:
        return "U-59,양호,숨겨진 파일 및 디렉터리가 존재하지 않거나 모두 정리됨"
    else:
        return "U-59,취약,다음 숨겨진 파일 및 디렉터리가 존재합니다."


# [U-60] 3.24 ssh 원격접속 허용
def check_u60():
    try:
        ssh_status = subprocess.check_output(['systemctl', 'is-active', 'sshd'],
stderr=subprocess.DEVNULL).decode().strip()
    except subprocess.CalledProcessError:
        ssh_status = 'inactive'
```

```
    try:
        telnet_rpm = subprocess.check_output(['rpm', '-q', 'telnet-server'], std
err=subprocess.DEVNULL).decode().strip()
        telnet_installed = telnet_rpm
    except subprocess.CalledProcessError:
        try:
            dpkg_output = subprocess.check_output(['dpkg', '-l'], stderr=subpr
ocess.DEVNULL).decode()
            telnet_installed = '\n'.join([line for line in dpkg_output.splitlines() if 't
elnetd' in line])
        except subprocess.CalledProcessError:
            telnet_installed = ''

    if ssh_status == 'active' and not telnet_installed:
        return "U-60,양호,원격 접속 시 SSH 프로토콜만 사용되고 Telnet은 비활성화
되어 있습니다."
    else:
        return (
            "U-60,취약,SSH 외에 Telnet이 설치되어 있거나 활성화되어 있을 수 있습니
다.\n"
            f"SSH 상태: {ssh_status}\n"
            f"Telnet 설치 여부: {telnet_installed or '없음'}"
        )


# [U-61] 3.25 ftp 서비스 확인
def check_u61():
    try:
        ftp_process = subprocess.check_output(
            "ps -ef | egrep 'vsftpd|proftp' | grep -v grep",
            shell=True,
            stderr=subprocess.DEVNULL
        ).decode().strip()
    except subprocess.CalledProcessError:
        ftp_process = ""

    if not ftp_process:
        return "U-61,양호,FTP 서비스가 비활성화되어 있습니다."
```

```
    else:
        return f"U-61,취약,FTP 서비스가 활성화되어 있습니다.\n{ftp_process}"


# [U-62] 3.26 ftp 계정 shell 제한
def check_u62():
    try:
        ftp_info = subprocess.check_output(
            "grep '^ftp:' /etc/passwd | awk -F: '{print $7}'",
            shell=True,
            stderr=subprocess.DEVNULL
        ).decode().strip()
    except subprocess.CalledProcessError:
        ftp_info = ""

    if ftp_info == "/bin/false":
        return "U-62,양호,ftp 계정에 로그인 불가 쉘이 설정되어 있습니다."
    else:
        return f"U-62,취약,ftp 계정에 로그인 가능한 쉘이 설정되어 있습니다. ({ftp_in
fo or '쉘 정보 없음'})"



# [U-63] 3.27 ftpusers 파일 소유자 및 권한 설정
def check_u63():
    results = []
    paths = ["/etc/ftpusers", "/etc/ftpd/ftpusers"]

    for file_path in paths:
        if os.path.isfile(file_path):
            stat_info = os.stat(file_path)
            owner = pwd.getpwuid(stat_info.st_uid).pw_name
            perm = oct(stat_info.st_mode & 0o777)[-3:]

            if owner == "root" and int(perm) <= 640:
                results.append(f"U-63,양호,{file_path}의 소유자와 권한이 적절합니
다.")
            else:
```

```
            results.append(f"U-63,취약,{file_path}의 소유자({owner}) 또는 권한
({perm})이 부적절합니다.")
        else:
            results.append(f"U-63,양호,{file_path} 파일이 존재하지 않습니다.")

    return "\n".join(results)


# [U-64] 3.28 ftpusers 파일 설정(FTP 서비스 root 계정 접근제한)
def check_u64():
    ftp_users_files = [
        "/etc/ftpusers",
        "/etc/ftpd/ftpusers",
        "/etc/vsftp/ftpusers",
        "/etc/vsftp/user_list",
        "/etc/vsftpd.ftpusers",
        "/etc/vsftpd.user_list"
    ]
    proftpd_conf = "/etc/proftpd.conf"

    root_allowed = False
    file_existed = False
    warnings = []

    for file_path in ftp_users_files:
        if os.path.isfile(file_path):
            file_existed = True
            try:
                with open(file_path, "r") as f:
                    lines = f.readlines()
                    for line in lines:
                        line = line.strip()
                        if line and not line.startswith("#") and line == "root":
                            warnings.append(f"[!] {file_path} 에서 root 계정 허용됨")
                            root_allowed = True
                            break
            except Exception:
                continue
```

```
    if os.path.isfile(proftpd_conf):
        file_existed = True
        try:
            with open(proftpd_conf, "r") as f:
                for line in f:
                    if line.strip().lower().startswith("rootlogin") and "on" in line.lowe
r():
                        warnings.append(f"[!] {proftpd_conf} 에서 RootLogin on 설정
됨")
                        root_allowed = True
                        break
        except Exception:
            pass

    if not file_existed:
        return "U-64,양호,FTP 관련 설정 파일이 존재하지 않거나, FTP 서비스 미설
치"
    elif not root_allowed:
        return "U-64,양호,FTP 서비스에서 root 계정의 접근이 제한됨"
    else:
        return "U-64,취약,FTP 서비스에서 root 계정의 접근이 허용됨 (접근 제한 필
요)\n" + "\n".join(warnings)


# [U-65] 3.29 at 서비스 권한 설정
def check_u65():
    at_allow_file = "/etc/at.allow"
    at_dir = "/var/spool/at"
    results = []

    # 1. at.allow 파일 확인
    if os.path.isfile(at_allow_file):
        if os.path.getsize(at_allow_file) == 0:
            results.append(f"U-65,취약,{at_allow_file} 파일이 비어있습니다. 일반사
용자 at 명령어 사용 가능")
        else:
            results.append(f"U-65,양호,{at_allow_file} 파일이 존재하며 일반사용자 a
```

```
t 명령어 사용이 제한됨")
    else:
        results.append(f"U-65,취약,{at_allow_file} 파일이 존재하지 않아 일반사용
자 at 명령어 사용 가능")

    # 2. /var/spool/at 디렉터리 권한 확인
    if os.path.isdir(at_dir):
        perm = oct(os.stat(at_dir).st_mode & 0o777)[-3:]
        if int(perm) <= 640:
            results.append(f"U-65,양호,{at_dir} 디렉터리 권한이 640 이하로 설정
됨")
        else:
            results.append(f"U-65,취약,{at_dir} 디렉터리 권한이 640 초과로 설정
됨")
    else:
        results.append(f"U-65,양호,{at_dir} 디렉터리가 존재하지 않음")

    return "\n".join(results)


# [U-66] 3.30 SNMP 서비스 구동 점검
def check_u66():
    try:
        output = subprocess.check_output(
            "ps -ef | grep snmpd | grep -v grep",
            shell=True,
            stderr=subprocess.DEVNULL
        ).decode().strip()
        if output:
            return "U-66,취약,SNMP 서비스가 구동 중입니다."
        else:
            return "U-66,양호,SNMP 서비스가 구동 중이지 않습니다."
    except subprocess.CalledProcessError:
        return "U-66,양호,SNMP 서비스가 구동 중이지 않습니다."


# [U-67] 3.31 SNMP 서비스 Community String의 복잡성 설정
def check_u67():
```

```
    conf_path = "/etc/snmp/snmpd.conf"

    if not os.path.isfile(conf_path):
        return "U-67,양호,SNMP 설정 파일이 존재하지 않습니다."

    try:
        with open(conf_path, "r") as f:
            lines = f.readlines()

        community_names = []
        for line in lines:
            if 'community' in line.lower():
                tokens = line.strip().split()
                if len(tokens) >= 2:
                    community_names.append(tokens[1].lower())

        if any(name in ["public", "private"] for name in community_names):
            return "U-67,취약,SNMP Community 문자열에 'public' 또는 'private'이
포함되어 있습니다."
        else:
            return "U-67,양호,SNMP Community 문자열이 복잡하게 설정되어 있습니
다."

    except Exception:
        return "U-67,오류,SNMP 설정 파일을 읽는 중 오류가 발생했습니다."


# [U-68] 3.32 로그온 시 경고 메시지 제공
def check_u68():
    results = []

    def check_message_file(file_path, desc):
        if not os.path.isfile(file_path):
            results.append(f"U-68,취약,{desc} 파일이 존재하지 않습니다.")
        elif os.path.getsize(file_path) == 0:
            results.append(f"U-68,취약,{desc} 파일에 경고 메시지가 설정되어 있지
않습니다.")
        else:
```

```
            results.append(f"U-68,양호,{desc} 파일에 경고 메시지가 설정되어 있습
니다.")

    def check_vsftpd_banner():
        conf_path = "/etc/vsftpd/vsftpd.conf"
        if not os.path.isfile(conf_path):
            results.append("U-68,취약,vsftpd 설정 파일이 존재하지 않습니다.")
        else:
            try:
                with open(conf_path, "r") as f:
                    if any(line.strip().startswith("ftpd_banner=") for line in f):
                        results.append("U-68,양호,vsftpd 설정에 FTP 경고 메시지가 설
정되어 있습니다.")
                    else:
                        results.append("U-68,취약,vsftpd 설정에 FTP 경고 메시지가 설
정되어 있지 않습니다.")
            except Exception:
                results.append("U-68,취약,vsftpd 설정 파일을 읽는 중 오류 발생")

    def check_sendmail_banner():
        conf_path = "/etc/mail/sendmail.cf"
        if not os.path.isfile(conf_path):
            results.append("U-68,취약,sendmail 설정 파일이 존재하지 않습니다.")
        else:
            try:
                with open(conf_path, "r") as f:
                    if any(line.strip().startswith("O SmtpGreetingMessage=") for lin
e in f):
                        results.append("U-68,양호,sendmail 설정에 SMTP 경고 메시지
가 설정되어 있습니다.")
                    else:
                        results.append("U-68,취약,sendmail 설정에 SMTP 경고 메시지
가 설정되어 있지 않습니다.")
            except Exception:
                results.append("U-68,취약,sendmail 설정 파일을 읽는 중 오류 발생")

    def check_named_conf():
        conf_path = "/etc/named.conf"
```

```
        if not os.path.isfile(conf_path):
            results.append("U-68,취약,named 설정 파일이 존재하지 않습니다.")
        else:
            results.append("U-68,양호,named 설정 파일이 존재합니다.")

    check_message_file("/etc/motd", "서버 로그인 메시지 (/etc/motd)")
    check_message_file("/etc/issue.net", "Telnet 배너 메시지 (/etc/issue.ne
t)")
    check_vsftpd_banner()
    check_sendmail_banner()
    check_named_conf()

    return "\n".join(results)


# [U-69] 3.33 NFS 설정파일 접근권한
def check_u69():
    file_path = "/etc/exports"

    if not os.path.isfile(file_path):
        return "U-69,취약,NFS 설정파일이 존재하지 않습니다."

    stat_info = os.stat(file_path)
    owner = pwd.getpwuid(stat_info.st_uid).pw_name
    perm_num = oct(stat_info.st_mode & 0o777)[-3:]

    if owner == "root" and int(perm_num) <= 644:
        return "U-69,양호,NFS 설정파일 소유자가 root이며 권한이 644 이하로 설정
되어 있습니다."
    else:
        return "U-69,취약,NFS 설정파일 소유자 또는 권한이 적절하지 않습니다."


# [U-70] 3.34 expn, vrfy 명령어 제한
def check_u70():
    file_path = "/etc/mail/sendmail.cf"

    if not os.path.isfile(file_path):
```

```
        return "U-70,양호,sendmail 설정 파일이 존재하지 않습니다."

    try:
        with open(file_path, "r") as f:
            content = f.read()
        has_noexpn = re.search(r'^(O )?PrivacyOptions.*noexpn', content, re.
MULTILINE)
        has_novrfy = re.search(r'^(O )?PrivacyOptions.*novrfy', content, re.M
ULTILINE)

        if has_noexpn and has_novrfy:
            return "U-70,양호,sendmail 설정에 noexpn, novrfy 옵션이 설정되어 있습
니다."
        else:
            return "U-70,취약,sendmail 설정에 noexpn, novrfy 옵션이 설정되어 있지
않습니다."

    except Exception:
        return "U-70,오류,sendmail 설정 파일을 읽는 중 오류가 발생했습니다."


# [U-71] 3.35 Apache 웹 서비스 정보 숨김
def check_u71():
    conf_file = "/etc/httpd/conf/httpd.conf"
    results = []

    if not os.path.isfile(conf_file):
        return "U-71,취약,Apache 설정 파일이 존재하지 않습니다."

    try:
        with open(conf_file, "r") as f:
            lines = f.readlines()

        server_tokens_ok = any(re.match(r'^ServerTokens\s+Prod', line.strip())
for line in lines)
        server_signature_ok = any(re.match(r'^ServerSignature\s+Off', line.stri
p()) for line in lines)
```

```
        if server_tokens_ok:
            results.append("U-71,양호,ServerTokens가 Prod로 설정되어 있습니다.")
        else:
            results.append("U-71,취약,ServerTokens가 Prod로 설정되어 있지 않습
니다.")

        if server_signature_ok:
            results.append("U-71,양호,ServerSignature가 Off로 설정되어 있습니
다.")
        else:
            results.append("U-71,취약,ServerSignature가 Off로 설정되어 있지 않습
니다.")

        return "\n".join(results)

    except Exception:
        return "U-71,오류,Apache 설정 파일을 읽는 중 오류가 발생했습니다."


# [U-72] 5.2 정책에 따른 시스템 로깅 설정
def check_u72():
    conf_file = "/etc/syslog.conf"
    results = []

    if not os.path.isfile(conf_file):
        return "U-72,취약,syslog 설정 파일이 존재하지 않습니다."

    try:
        with open(conf_file, "r") as f:
            content = f.read()

        patterns = [
            (r'^\*.info;mail.none;authpriv.none;cron.none\s+/var/log/messages',
"기본 메시지 로그"),
            (r'^authpriv\.\*\s+/var/log/secure', "인증 관련 로그"),
            (r'^mail\.\*\s+/var/log/maillog', "메일 로그"),
            (r'^cron\.\*\s+/var/log/cron', "크론 로그"),
            (r'^\*.alert\s+/dev/console', "alert 로그"),
```

```
            (r'^\*.emerg\s+\*', "emerg 로그")
        ]

        for pattern, desc in patterns:
            if re.search(pattern, content, re.MULTILINE):
                results.append(f"U-72,양호,{desc} 설정이 존재합니다.")
            else:
                results.append(f"U-72,취약,{desc} 설정이 없습니다.")

        return "\n".join(results)

    except Exception:
        return "U-72,오류,syslog 설정 파일을 읽는 중 오류가 발생했습니다."



def run_all_checks():
    checks = [
        check_u01, check_u02, check_u03, check_u04, check_u05,
        check_u06, check_u07, check_u08, check_u09, check_u10,
        check_u11, check_u12, check_u13, check_u14, check_u15,
        check_u16, check_u17, check_u18, check_u19, check_u20,
        check_u21, check_u22, check_u23, check_u24, check_u25,
        check_u26, check_u27, check_u28, check_u29, check_u30,
        check_u31, check_u32, check_u33, check_u34, check_u35,
        check_u36, check_u37, check_u38, check_u39, check_u40,
        check_u41, check_u42, check_u43, check_u44, check_u45,
        check_u46, check_u47, check_u48, check_u49, check_u50,
        check_u51, check_u52, check_u53, check_u54, check_u55,
        check_u56, check_u57, check_u58, check_u59, check_u60,
        check_u61, check_u62, check_u63, check_u64, check_u65,
        check_u66, check_u67, check_u68, check_u69, check_u70,
        check_u71, check_u72
    ]

    results = []
    for check_func in checks:
        try:
```

```
            results.append(check_func())
        except Exception as e:
            results.append(f"{check_func.__name__},오류,{str(e)}")
    return results


if __name__ == "__main__":
    results = run_all_checks()

    # 콘솔 출력
    for result in results:
        print(result)

    # CSV 저장
    with open("rocky_result.csv", "w", newline='', encoding="utf-8-sig") as f:
        writer = csv.writer(f)
        writer.writerow(["항목", "결과", "설명"])  # CSV 헤더
        for line in results:
            parts = line.split(",", 2)
            if len(parts) == 3:
                writer.writerow(parts)
            else:
                writer.writerow([line, "", ""])  # 예외적인 형식도 기록
```