

**32230324 김가람  
32210138 고대현  
32221114 김준범**

# RansomShiled

RansomWAR | BLUE01



# CONTENTS

01

팀명, 역할 소개



02

주요 모듈 구성



03

핵심 방어 기능



04

개선사항





# Team RansomShield

Real-time Defense Against Ransomware

## 고대현 (팀장)

- 프로젝트 총괄 및 일정 관리
- 핵심 방어 로직 코드 작성 및 보완 (엔트로피, 레이트 리밋 등)
- 최종 보고서 작성

## 김가람

read, unlink, open 행위 관련 코드 구현  
(rate limit, 화이트리스트 확장자, 확장자/매

직넘버 일치 여부 검사 등)

- write 쿨다운 기능 추가, 스냅샷 복구 기능 추가, I/O 리듬 분석 기능 추가

- 코드 보완 : 엔트로피 로직 보완, write rate limit 기능 보완(파일별 → 전체) 등

- 일정, 회의록, Github 관리

- 프로젝트 발표 담당

## 김준범

- 기능 구현 코드 작성 및 보완 (로그, 유틸리티 등)

- 발표 자료(PPT) 제작

# 주요 모듈 구성

**myfs\_write (쓰기 행위 감시)**

엔트로피 계산 및 암호화 탐지

쓰기 빈도감시

파일 크기 60% 이상 감소 시 차단

**myfs\_rename (이름 변경 제어)**

확장자-매직넘버 일치 여부 검사  
rename 횟수/속도 제한

**myfs\_read (읽기 제어)**

프로세스별 대량 읽기 감시

# 주요 모듈 구성

**myfs\_create (파일 생성 감시)**

화이트리스트 확장자만 허용

랜섬노트명(README, DECRYPT 등) 차단

**myfs\_unlink (삭제 제어)**

10초 내 2회 이상 삭제 차단

대량 삭제 탐지 로그 기록

**myfs\_open(파일 열기)**

확장자 및 매직넘버 일치 여부 검증

쓰기용 파일 상태 초기화

🔍 **엔트로피 기반 암호화  
탐지**

**Rate Limit &  
I/O리듬분석**

**핵심 방어 기능**

🔍 **스냅샷 백업 및 복구**

**위장공격 차단**

# 엔트로피 기반 암호화 탐지

쓰기 시 엔트로피 계산 → 암호화 시도 실시간 감지

엔트로피 7.0 초과 시 차단 및 일정시간 쓰기 금지

엔트로피 6.5 초과 시 경고

로그 예시 -> [2025-11-24T19:35:12+0900] [BLOCKED]  
uid=1000 pid=4521 action=write path="/home/user/workspace/target/report.docx"  
reason="excessive-high-entropy"  
extra="entropy = 7.68 size=4096 offset=0 cooldown=10"

```
static double calculate_entropy(const char *buf, size_t size) {
    if (size == 0) return 0.0;

    // 빈도 수 측정
    // byte별로 몇 번 등장하였는지 count한다
    int counts[256] = {0};
    for (size_t i = 0; i < size; i++) {
        counts[(unsigned char)buf[i]]++;
    }

    double entropy = 0.0;
    for (int i = 0; i < 256; i++) {
        if (counts[i] > 0) {
            // 확률 계산: P(x) = (해당 byte 등장 횟수) / (전체 크기)
            double p = (double)counts[i] / size;

            entropy -= p * log2(p);
        }
    }
    return entropy;
}
```

## 쓰기 시 엔트로피 계산

```
double entropy = calculate_entropy(buf, size);
if (entropy > HIGH_ENTROPY_HARD_BLOCK) {
    if (state) {
        // entropy 7.5 초과 -> 암호화를 시도하는 것으로 판단하여 즉시 차단하고 차단 해제 대기시간 적용
        pthread_mutex_lock(&state_mutex);
        state->blocked = 1;
        state->blocked_until = now_s + FILE_BLOCK_COOLDOWN_SEC;
        pthread_mutex_unlock(&state_mutex);
    }
    log_line("WRITE", path, "BLOCKED", "excessive-high-entropy", "entropy=%.2f size=%zu offset=%ld cooldown=%ds", entropy, size, (size_t)offset, FILE_BLOCK_COOLDOWN_SEC);
    return -EPERM;
} else if (entropy > HIGH_ENTROPY_THRESHOLD) {
    // entropy가 6.5 초과 -> 의심스러운 행위이지만 압축 파일일 수도 있으므로 로그를 남긴다.
    log_line("WRITE", path, "FLAG", "high-entropy", "entropy=%.2f", entropy);
}
```

## 엔트로피 7.0 초과 시 차단 및 일정시간 쓰기 금지

// 엔트로피 6.5 초과 시 경고

# Rate Limit

**Read: PID별 읽기량 제한 (10초 50MB 초과 시 차단)**

**Unlink: 일정 시간 내 삭제 횟수 초과 시 차단**

**Rename: 파일당 rename 횟수 / 빈도 제한**

**Write: 5초 내 3회이상 시도할 경우 감지 및 차단**

로그 예시 -> [2025-11-24T19:35:12+0900] [BLOCKED]  
uid=1000 pid=4521 action=unlink path="/home/user/workspace/target/test.txt"  
reason="rate-limit"

extra="windows=10s max=3 count=4"

# I/O리듬분석

I/O 요청 간 간격 분석

일정한 리듬 탐지 시 비정상 행위로 인식

Rate Limit 기능 우회 방지

로그 예시 -> [2025-11-24T19:35:12+0900] [BLOCKED]  
uid=1000 pid=4521 action=write path="/home/user/workspace/target/test.txt"  
reason="IAT-low-jitter"  
extra="mean=0.0534 stddev=0.00172 samples=12"

```

// 읽기량 누적 / 임계 초과 검사
if (rs && res > 0) {
    int just_blocked = 0;
    pthread_mutex_lock(&g_read_lock);
    time_t now = time(NULL);

    // 윈도우 리셋
    if (difftime(now, rs->win_start) >= READ_WINDOW_SEC) {
        rs->win_start = now;
        rs->bytes = 0;
        rs->blocked = 0;
    }
    rs->bytes += (size_t)res; // 이번 호출에서 읽은 양 res만큼 누적

    // 허용량 초과하면 차단 상태로 전환
    if (rs->bytes > MAX_READ_BYTES_PER_WINDOW) {
        rs->blocked = 1;
        just_blocked = 1;
    }
    pthread_mutex_unlock(&g_read_lock);
}

// 방금 차단 상태 되었으면 FLAG 로그
if (just_blocked) {
    log_line("READ", path, "FLAG", "rate-limit-read-tripped",
            "pid=%d bytes=%zu limit=%zu window=%ds",
            (int)cur_pid, rs->bytes, (size_t)MAX_READ_BYTES_PER_WINDOW, READ_WINDOW_SEC);
}

```

## Read: PID별 읽기량 제한 (10초 50MB 초과 시 차단)

```

// UNLINK 레이트 리밋 함수
// 현재 시간이 기존 윈도우 시작 시간보다 UNLINK_WINDOW_SEC 이상 지났으면 윈도우를 리셋하고 카운터 다시 시작
// 삭제 횟수를 1 증가시키고, 임계치를 초과하면 1 반환
static int unlink_rate_limit_exceeded(int *out_count_in_window) {
    time_t now = time(NULL);

    // 윈도우가 아직 시작되지 않았거나 윈도우 시간이 지났으면 리셋
    if (rl_window_start == 0 || difftime(now, rl_window_start) >= UNLINK_WINDOW_SEC) {
        rl_window_start = now;
        rl_unlink_count = 0;
    }

    rl_unlink_count++; // 삭제 시도 1 증가

    // 호출자가 현재 윈도우 내의 삭제 횟수를 알고 싶으면 out_count_in_window에 저장
    if (out_count_in_window) *out_count_in_window = rl_unlink_count;

    // 최대 허용 횟수를 초과했는지 여부 반환
    return (rl_unlink_count > MAX_UNLINK_PER_WINDOW);
}

```

## Unlink: 일정 시간 내 삭제 횟수 초과 시 차단

```
// 일정 시간(1초) 내 반복 rename → rename flood 공격으로 간주
if (info->last_time != 0 && now - info->last_time < 1 /* RENAME_INTERVAL */) {
    pthread_mutex_unlock(&rename_lock);
    log_line("RENAME", relto, "BLOCKED", "Rename flood detected (per file)", NULL);
    return -EPERM;
}

// 파일별 rename 횟수 5회 초과 시 차단
if (info->count >= 5) {
    pthread_mutex_unlock(&rename_lock);
    log_line("RENAME", relto, "BLOCKED", "Rename limit exceeded (per file)", NULL);
    return -EPERM;
}
```

## Rename: 파일당 rename 횟수 // 빈도 제한

```

// WRITE 레이트 리밋 함수
// 3초에 3회 이상 write 시도 시 차단
static int write_rate_limit_exceeded(void) {
    time_t now = time(NULL);

    pthread_mutex_lock(&write_freq_lock);

    int i = 0;
    while (i < write_ts_count) {
        if (difftime(now, write_timestamps[i]) > WRITE_WINDOW_SEC) {
            // 오래된 것은 배열 마지막 요소로 엮어쓰고 개수 줄이기
            write_timestamps[i] = write_timestamps[write_ts_count - 1];
            write_ts_count--;
        }
        else {
            i++;
        }
    }
}

```

```

// 현재 윈도우 내 write 시도 개수 확인
// 5초에 3회 이상 -> 2개까지 허용, 3번째부터 차단
if (write_ts_count >= (MAX_WRITES_IN_WINDOW - 1)) {
    pthread_mutex_unlock(&write_freq_lock);
    return 1;
}

// 허용되는 경우, 이번 시도 타임스탬프 추가
if (write_ts_count < (int)(sizeof(write_timestamps)/sizeof(write_timestamps[0]))) {
    write_timestamps[write_ts_count++] = now;
}

pthread_mutex_unlock(&write_freq_lock);
return 0;
}

```

## Write: 5초 내 3회이상 시도할 경우 감지 및 차단

```

// 표본 충분하면 Low Jitter 판단
if (st->count >= IAT_MIN_SAMPLES && st->m2 > 0.0) {
    // 표준분산 : m2 / (n-1)
    double var = st->m2 / (st->count - 1);
    if (var < 0.0) var = 0.0;
    stddev = sqrt(var);
    count = st->count;

    double mean_iat = st->mean;
    // 상대적 지터 : 표준편차 / 평균 (평균 대비 변동 폭이 어느 정도인지)
    double rel_jitter = (mean_iat > 0.0) ? (stddev / mean_iat) : 1e9;

    // 절대 표준편차가 너무 작아도 의심
    // 평균 간격이 어느 정도(0.5초 이상)인데도 상대적 지터 작으면 느린 공격으로 의심
    if (stddev < IAT_LOW_JITTER_STDDEV || (mean_iat > 0.5 && rel_jitter < IAT_REL_JITTER_THRESHOLD)) {
        st->flagged = 1; // 이 actor를 low jitter 패턴으로 확정
    }
} else {
    // 표본 부족하면 0으로 처리
    stddev = 0.0;
    count = st->count;
}

```

## I/O 리듬 분석 (Rate Limit 기능 우회 방지)

# 스냅샷 백업 및 복구

**첫 쓰기 전에 자동 백업 (`$HOME/.snapshots/`)**

**스냅샷 백업본을 통해 손상된 파일 원본 복구 가능**

**복구 후 즉시 write 시도 방지 → 끝다운 기능 활용**

```
로그 예시 -> [2025-11-24T19:35:12+0900] [ALLOW]
uid=1000 pid=4521 action=SNAPSHOT
path="/home/user/workspace/target/test.docx"
reason="backup-ok"
extra="dst = ..."
```

```
// 소유 외에는 백업 폴더에 접근할 수 없도록 차단
if (access(backup_dir_path, F_OK) == -1) {
    if (mkdir(backup_dir_path, 0700) == -1) {
        log_line("SNAPSHOT", path, "FAIL", "Cannot create backup dir", "errno=%d", errno);
        return -EIO;
    }
}
```

**백업 디렉터리 자동생성 //  
소유자만 접근 가능하도록 0700 권한 적용**

```
// 실제 복사  
char buf[4096];  
ssize_t n;  
while ((n = read(src_fd, buf, sizeof(buf))) > 0) {  
    if (write(dst_fd, buf, n) != n) {  
        close(src_fd);  
        close(dst_fd);  
        log_line("SNAPSHOT", path, "FAIL", "write error", "errno=%d", errno);  
        return -EIO;  
    }  
}
```

**원본파일 내용을 읽어 백업디렉터리에 복사**  
**// 쓰기 실패 시 즉시 차단 및 로그 기록**

```

// 스냅샷 열기
int src_fd = open(snapshot_path, O_RDONLY);
if (src_fd == -1) {
    log_line("RESTORE", fuse_path, "FAIL", "open-snapshot-failed",
             "errno=%d path=\"%s\"", errno, snapshot_path);
    return -errno;
}

// 원본 파일은 base_fd 기준 relpath에 직접 써서 복구 (FUSE 레이어 우회)
int dst_fd = openat(base_fd, relpath, O_WRONLY | O_TRUNC | O_CREAT, 0600);
if (dst_fd == -1) {
    int e = errno;
    close(src_fd);
    log_line("RESTORE", fuse_path, "FAIL", "open-dst-failed",
             "errno=%d relpath=\"%s\"", e, relpath);
    return -e;
}

```

```

// 실제 복사
char buf[4096];
ssize_t n;
int copy_err = 0;

while ((n = read(src_fd, buf, sizeof(buf))) > 0) {
    ssize_t w = write(dst_fd, buf, n);
    if (w != n) {
        copy_err = (w < 0) ? errno : EIO;
        break;
    }
}
if (n < 0 && copy_err == 0) {
    copy_err = errno;
}

```

## 스냅샷 백업본을 통해 손상된 파일 원본 복구

# 위장공격 차단

열기/생성/이름변경 시 화이트리스트 기반 검사

매직넘버 불일치 시 위장 탐지 후 차단

랜섬노트 이름(readme, decrypt) 생성 차단

로그 예시 → [2025-11-24T19:35:12+0900] [BLOCKED]

```
uid=1000 pid=4521 action=rename path="/home/user/workspace/target/test.txt"
reason="Deep Magic number mismatch (16-byte signature failed for new
extension)"
```

```
// 화이트리스트 확장자인지 확인하는 함수

static int is_sensitive_ext(const char *ext) {
    if (!ext || !*ext) return 0;
    for (size_t i=0;i<sizeof(SENSITIVE_EXTS)/sizeof(SENSITIVE_EXTS[0]);i++) {
        if (strcmp(ext, SENSITIVE_EXTS[i]) == 0) return 1;
    }
    return 0;
}
```

## 화이트리스트 기반 검사 로직

```
// 확장자 화이트리스트 & 랜섬노트 패턴
// 보호해야 할 민감/중요 데이터 확장자 목록 (전부 소문자)
static const char *SENSITIVE_EXTS[] = {
    "doc", "docx", "docb", "docm", "dot", "dotm", "dotx",
    "xls", "xlsx", "xlsm", "xlsb", "xlw", "xlt", "xlm", "xlc", "xltx", "xltm",
    "ppt", "pptx", "pptm", "pot", "pps", "ppsm", "ppsx", "ppam", "potx", "potm",
    "pst", "ost", "msg", "emi", "edb", "vsd", "vsdx", "txt", "csv", "rtf", "123",
    "wks", "wk1", "pdf", "dwg", "onectoc2", "snt", "hwp", "602", "sxi", "sti",
    "sldx", "sldm", "vdi", "vmdk", "vmx", "gpg", "aes", "arc", "paq", "bz2",
    "tbk", "bak", "tar", "tgz", "gz", "7z", "rar", "zip", "backup", "iso", "vcd",
    "jpeg", "jpg", "bmp", "png", "gif", "raw", "cgm", "tif", "tiff", "net", "psd",
    "ai", "svg", "djvu", "m4u", "m3u", "mid", "wma", "flv", "3g2", "mkv", "3gp",
    "mp4", "mov", "avi", "ASF", "mpeg", "vob", "mpg", "wmv", "fla", "swf", "wav",
    "mp3", "sh", "class", "jar", "java", "rb", "asp", "php", "jsp", "brd", "sch",
    "dch", "dip", "pl", "vb", "vbs", "ps1", "bat", "cmd", "js", "asm", "h", "pas",
    "cpp", "c", "cs", "suo", "sln", "ldf", "mdf", "ibd", "myi", "myd", "frm",
    "odb", "dbf", "db", "mdb", "accdb", "sql", "sqlitedb", "sqlite3", "asc",
    "lay6", "lay", "mml", "sxm", "otg", "odg", "uop", "std", "sxd", "otp", "odp",
    "wb2", "slk", "dif", "stc", "sxc", "ots", "ods", "3dm", "max", "3ds", "uot",
    "stw", "sxw", "ott", "odt", "pem", "p12", "csr", "crt", "key", "pfx", "der", "tmp", "bin",
    "py", "rs", "go", "lua", "ts", "dll", "html", "exe", "json", "log", "old", "sqlite", "hwpx"
};
```

## 화이트리스트 목록

```

// 확장자와 매직 넘버 일치하는지 확인하는 함수
// ext에 따라 각 파일 포맷의 매직 넘버를 비교
// 매직 넘버를 알 수 없는 확장자는 검사를 하지 않고 허용(1 반환)
// n <= 0 이거나 ext가 없으면 판단 불가 -> 허용(1)
static int magic_ok_for_ext(const char *ext, const unsigned char *h, ssize_t n) {
    if (!ext || !*ext || n<=0) return 1; // 정보 부족 -> 판단 불가 -> 허용

    // 주요 포맷들에 대한 매직 넘버 체크
    if (!strcmp(ext,"pdf")) return starts_with(h,n,(const unsigned char*)"%PDF",4);
    if (!strcmp(ext,"png")) return starts_with(h,n,(const unsigned char*)"x89PNG",4);
    if (!strcmp(ext,"jpg") || !strcmp(ext,"jpeg")) return (n>=2 && h[0]==0xFF && h[1]==0xD8);
    if (!strcmp(ext,"gif")) return starts_with(h,n,(const unsigned char*)"GIF",3);
    if (!strcmp(ext,"tif") || !strcmp(ext,"tiff"))
        return (starts_with(h,n,(const unsigned char*)"II*\0",4) || starts_with(h,n,(const unsigned char*)"MM\0*",4));
    if (!strcmp(ext,"psd")) return starts_with(h,n,(const unsigned char*)"8BPS",4);

    // ZIP / OOXML 계열
    if (!strcmp(ext,"zip")||!strcmp(ext,"docx")||!strcmp(ext,"xlsx")||!strcmp(ext,"pptx")||
        !strcmp(ext,"xltx")||!strcmp(ext,"xltm")||!strcmp(ext,"potx")||!strcmp(ext,"potm")||
        !strcmp(ext,"ppsx")||!strcmp(ext,"ppsm")||!strcmp(ext,"sldx")||!strcmp(ext,"sldm"))
        return starts_with(h,n,(const unsigned char*)"PK\x03\x04",4);

```

```

// OLE 계열 (doc/xls/ppt/하나워드 등)
if (!strcmp(ext,"doc")||!strcmp(ext,"xls")||!strcmp(ext,"ppt")||!strcmp(ext,"vsd")||
    !strcmp(ext,"msg")||!strcmp(ext,"hwp"))
    return starts_with(h,n,(const unsigned char*)"xD0\xCF\x11\xE0\xA1\xB1\x1A\xE1",8);

// 압축 포맷
if (!strcmp(ext,"gz")||!strcmp(ext,"tgz")) return (n>=2 && h[0]==0x1F && h[1]==0x8B);
if (!strcmp(ext,"bz2")) return starts_with(h,n,(const unsigned char*)"BZh",3);
if (!strcmp(ext,"7z")) return starts_with(h,n,(const unsigned char*)"7z\xBC\xAF\x27\x1C",6);
if (!strcmp(ext,"rar"))
    return (starts_with(h,n,(const unsigned char*)"Rar!\x1A\x07\x00",7) ||
            starts_with(h,n,(const unsigned char*)"Rar!\x1A\x07\x01\x00",8));

// sqlite 계열
if (!strcmp(ext,"sqlite3")||!strcmp(ext,"sqlitedb"))
    return starts_with(h,n,(const unsigned char*)"SQLite format 3",16);

return 1; // 매직 넘버를 모르는 확장자는 검사하지 않음
}

```

# 매직넘버 불일치 시 위장 탐지 후 차단

```
// 경로 이름이 랜섬노트 패턴을 포함하는지 검사하는 함수
// 전체 경로를 소문자로 변환 후 부분 문자열 검색
static int is_ransom_note(const char *path) {
    char lower_path[PATH_MAX];
    to_lower_str(path, lower_path, sizeof(lower_path));
    for (const char **n = ransom_note_names; *n; n++) {
        if (strstr(lower_path, *n) != NULL) {
            return 1; // 패턴이 하나라도 포함되면 랜섬노트로 판단
        }
    }
    return 0;
}
```

## (랜섬노트 이름 목록)

```
// 랜섬노트 이름에 자주 등장하는 키워드 패턴
static const char *ransom_note_names[] = {
    "readme", "decrypt", "how_to", NULL
};
```

# 랜섬노트 이름 생성 차단

# 개선사항



편집기 호환성 개선



스냅샷 ON /  
OFF 제어

# 개선 전

**편집기 사용 시 .swp, .goutputstream 등의 임시 확장자 파일이 자동 생성**

**-> 이 확장자들이 화이트리스트에 포함되지 않아 차단 오류가 발생**

## 로그 1

```
623 [2025-12-01T11:57:24+0900] [BLOCKED]
624 uid=1000 pid=5121 action=CREATE path=".goutputstream-QCMWG3"
625 reason="File extension not in whitelist policy (e.g., no extension)"
```

## 로그 2

```
1824 [2025-12-01T12:05:40+0900] [BLOCKED]
1825 uid=1000 pid=2824 action=CREATE path=".test_69.txt.swp"
1826 reason="File extension not in whitelist policy (e.g., no extension)"

1827
1828 [2025-12-01T12:05:40+0900] [BLOCKED]
1829 uid=1000 pid=2824 action=CREATE path=".test_69.txt.swp"
1830 reason="File extension not in whitelist policy (e.g., no extension)"
```

# 편집기 호환성 개선



**편집기 사용 시 .swp, .goutputstream 등의 임시 파일 생성**

→ 화이트리스트 검사 // rate limit 정책에 의해 차단되는 문제 발생



**임시/백업 파일 // PID 기반 프로세스 식별 예외 처리 로직 추가**

→ 편집기 사용 시 정상적인 쓰기·수정 가능

```

// 편집기(vim, gedit 등)가 만드는 임시/백업 파일인지 확인
// 예외적으로 화이트리스트/매직 검사에서 허용하기 위함
static int is_editor_temp_name(const char *path) {
    // basename만 빼어내기
    const char *base = strrchr(path, '/');
    base = base ? base + 1 : path;

    if (base[0] == '\0')
        return 0;

    // gedit / glib 가 만드는 임시 파일: .goutputstream-XXXX
    if (strncmp(base, ".goutputstream-", 15) == 0)
        return 1;
}

```

```

// vim 백업 파일: xxx~, xxx.cp~ 같은 것들
size_t len = strlen(base);
if (len > 0 && base[len - 1] == '~') {
    return 1;
}

// vim 스왑 파일: .xxx.swp, .xxx.swo 등 (있으면 편하게 허용)
if (len >= 4) {
    if (strcmp(base + len - 4, ".swp") == 0 ||
        strcmp(base + len - 4, ".swo") == 0 ||
        strcmp(base + len - 4, ".swx") == 0) {
        return 1;
    }
}

return 0;
}

```

## 편집기 임시파일 예외 처리

# 스냅샷 ON / OFF 제어



**기존에는 스냅샷 기능이 항상 ON 상태**

**-> 문서 편집 중에도 불필요한 백업이 지속적으로 생성**



**~/workspace/.snapshot 파일 생성**

**-> echo 0 > .snapshot 으로 비활성화 (기본값은 1)**

```
// ~/workspace/.snapshot_control 읽어서 g_snapshot_restore_enabled 갱신
static void refresh_snapshot_flag(void) {
    if (g_snapshot_ctrl_path[0] == '\0')
        return;

    int fd = open(g_snapshot_ctrl_path, O_RDONLY);
    if (fd == -1) {
        // 못 열면 기본 OFF
        g_snapshot_restore_enabled = 0;
        return;
    }

    char ch = 0;
    ssize_t n = read(fd, &ch, 1);
    close(fd);

    if (n == 1 && ch == '1')
        g_snapshot_restore_enabled = 1;
    else
        g_snapshot_restore_enabled = 0;
}
```

**스냅샷 복구기능 ON // OFF**

## (스냅샷 복구기능 ON // OFF 제어변수)

```
// 스냅샷 복구 on/off
static int g_snapshot_restore_enabled = 1; // default 값은 1
static char g_snapshot_ctrl_path[PATH_MAX] = {0}; // 제어 파일 경로
#define SNAPSHOT_CTRL_FILE ".snapshot_control"
```



**감사합니다.**