ShadowTrace

File Monitoring & Anomaly Detection System

Version 1.0 - Complete Technical Documentation

Created by: Mohamed Ehab

Table of Contents

1. Introduction

1.1 Overview

ShadowTrace is an advanced file monitoring and anomaly detection system designed to provide real-time surveillance of file system activities. By combining the high performance of C with the flexibility of Python, ShadowTrace offers a robust solution for detecting suspicious file operations, potential security threats, and unauthorized modifications.

1.2 Key Features

- High-performance file scanning using native C library
- Real-time monitoring with configurable scan intervals
- Intelligent anomaly detection algorithms
- Behavioral profiling of file modifications
- Automated email alerts and detailed reports
- Cross-platform compatibility (Windows focus)
- Minimal resource usage and efficient memory management

1.3 Use Cases

- Security monitoring of critical directories
- Ransomware detection through file behavior analysis
- Compliance auditing and file change tracking
- Development environment monitoring
- Backup verification and integrity checking
- Incident response and forensic analysis

2. System Architecture

2.1 Component Overview

ShadowTrace employs a modular architecture with clear separation of concerns:

2.2 Core Components

Component	Description
shadowtrace.py	Main entry point providing CLI interface and user interaction
file_monitor.py	Core monitoring engine with Python-C integration and anomaly detection
file_scanner.c/h	High-performance C library for directory traversal and metadata collection
email_reporter.py	Email notification system for automated reporting
libcheckpass.so	Compiled shared library for fast file system operations

2.3 Data Flow

1. User initiates monitoring through shadowtrace.py
2. file_monitor.py calls C library via ctypes
3. file_scanner.c performs rapid directory scanning
4. Metadata returned to Python for analysis
5. Snapshots created and compared for changes
6. Anomaly detection algorithms evaluate patterns
7. Results displayed in terminal and optionally emailed

3. Installation Guide

3.1 Prerequisites

Requirement	Specification
Operating System	Windows 7/8/10/11
Python	Version 3.8 or higher
C Compiler	GCC (MinGW) or MSVC
RAM	Minimum 256MB available

3.2 Step-by-Step Installation

Step 1: Clone the Repository

```
git clone https://github.com/mhd200722/shadowtrace.git
cd shadowtrace
```

Step 2: Install Python Dependencies

```
pip install pyfiglet termcolor
```

Step 3: Compile C Library (Windows with MinGW)

```
gcc -shared -o libcheckpass.so file_scanner.c -lkernel32
```

Step 4: Verify Installation

```
python shadowtrace.py
```

4. File Descriptions

4.1 shadowtrace.py (Main Entry Point)

Purpose: Provides the command-line interface and orchestrates the monitoring process.

Key Functions:

- Displays ASCII art banner using pyfiglet
- Collects user input (directory path, scan count, interval)
- Initiates monitoring through file_monitor.py
- Displays summary statistics
- Handles email report sending

4.2 file_monitor.py (Monitoring Engine)

Purpose: Core monitoring logic with anomaly detection capabilities.

Key Components:

Function	Description
scan_dir_py()	Interfaces with C library to scan directory
create_snapshot()	Creates dictionary mapping file paths to metadata
detect_anychanges()	Compares snapshots to identify added/deleted/modified files
update_behavior_profile()	Tracks modification counts and size history
detect_anomalies()	Identifies suspicious file behavior patterns
monitor_directory()	Main monitoring loop orchestrating all operations
filetime_to_datetime()	Converts Windows FILETIME to Python datetime

4.3 file_scanner.c/h (C Library)

Purpose: High-performance directory scanning using Windows API.

Key Structures:

- record: Contains file metadata (name, path, size, timestamps)
- Collections: Dynamic array of records with capacity management

Key Functions:

- Initialize(): Creates initial collection structure
- Scan_Directory(): Traverses directory and populates records
- add_record(): Adds file metadata to collection
- CleanUp(): Frees allocated memory
- filetime(): Converts FILETIME to long long integer

4.4 email_reporter.py (Email System)

Purpose: Generates and sends email reports via SMTP.

Key Functions:

- send_anomaly_report(): Main function to send comprehensive reports
- generate_report_text(): Formats monitoring results into readable text
- format_file_size(): Converts bytes to human-readable format

5. How to Run

5.1 Basic Execution

Execute the main script:

```
python shadowtrace.py
```

5.2 Interactive Prompts

Directory Path: Enter the full path to the directory you want to monitor
Example: C:\Users\YourName\Documents

Number of Scans: Specify how many times to scan the directory
Default: 10
Recommended: 5-20 for testing, 50-100 for monitoring

Scan Interval: Time in seconds between scans
Default: 60
Recommended: 30-300 depending on urgency

Email Report: Choose Y to send email report, N to skip
Requires email configuration

5.3 Example Session

```
Enter Your Path Directory: C:\Users\John\Documents
Enter Number Of Scans (Default: 10): 5
Enter Scan Interval in seconds (Default: 60): 30
Send email report after completion? [Y/N]: Y

Starting monitoring...
Scan 1/5 at 2026-02-15 10:30:00
No anomalies detected in this scan
Waiting 30 seconds until next scan...
```

6. Configuration

6.1 Anomaly Detection Parameters

Edit file_monitor.py to adjust sensitivity:

```
def detect_anomalies(snapshot, behavior_profile,
                mod_threshold=3,    # Files modified > 3 times
trigger alert
                growth_factor=2.0):  # Files growing > 2x trigger
alert
```

6.2 Email Configuration

Configure SMTP settings in email_reporter.py:

SMTP Server: smtp.gmail.com
Port: 587
Security: TLS
Authentication: App Password (recommended for Gmail)

Steps to get Gmail App Password:

1. Enable 2-Factor Authentication in Google Account
2. Go to Security → 2-Step Verification → App Passwords
3. Select "Mail" and your device
4. Copy the generated 16-character password
5. Use this password in the email_reporter.py login function

7. Anomaly Detection

7.1 Detection Algorithms

7.1.1 Frequent Modification Detection

Monitors how many times a file is modified during the monitoring period. Files modified more than the threshold (default: 3) are flagged as anomalies. This can indicate malicious activity such as ransomware encryption or data exfiltration.

7.1.2 Sudden Size Growth Detection

Tracks file size history and detects dramatic increases. If a file grows by more than the growth factor (default: 2.0x) between scans, it triggers an alert. This can indicate log flooding, data dumping, or file corruption attacks.

7.2 Behavioral Profiling

The system maintains a profile for each file containing:

- Modification count: Total number of changes detected
- Size history: List of file sizes over time
- Last modified timestamp: Most recent modification time

7.3 False Positive Reduction

The latest version eliminates false positives from the initial scan. Previously, all existing files were flagged as "new file anomalies" on the first run. This has been fixed to only detect truly anomalous behavior.

8. Email Reporting

8.1 Report Contents

Each email report includes:

- Report header with timestamp and directory path
- Total scans completed
- Security anomalies detected with descriptions
- File change summary (added, deleted, modified)
- Detailed file information (size, timestamps)
- Statistics summary
- Action recommendations

8.2 Report Format

Reports are formatted with:

- ASCII art borders for readability
- Emoji indicators for different event types
- Organized sections with clear headings
- Human-readable file sizes (B, KB, MB, GB)
- Formatted timestamps
- Color-coded severity levels (in console output)

9. Troubleshooting

FileNotFoundError: libcheckpass.so

Cause: C library not compiled or not in correct location

Solution: Compile the C library using: gcc -shared -o libcheckpass.so file_scanner.c -lkernel32

Email not sending

Cause: SMTP credentials incorrect or network issue

Solution: Verify email and app password in email_reporter.py. Check internet connection. For Gmail, ensure 2FA is enabled and app password is used.

Permission denied when scanning directory

Cause: Insufficient permissions to access directory

Solution: Run as administrator or choose a directory with appropriate permissions

All files shown as anomalies on first scan

Cause: Bug in older version (now fixed)

Solution: Update to latest version of file_monitor.py from repository

Import error for ctypes or other modules

Cause: Python dependencies not installed

Solution: Install required packages: pip install pyfiglet termcolor

10. API Reference

10.1 Python Functions

scan_dir_py(directory_path)

Parameters: directory_path (str): Path to scan

Returns: list: File records with metadata

Description: Calls C library to scan directory and returns file information

create_snapshot(file_records)

Parameters: file_records (list): List of file info dictionaries

Returns: dict: Mapping of file paths to metadata

Description: Creates a snapshot dictionary for change detection

detect_anychanges(old_snapshot, new_snapshot)

Parameters: old_snapshot (dict), new_snapshot (dict): Snapshot dictionaries

Returns: tuple: (added, deleted, modified) file lists

Description: Compares two snapshots to identify file changes

monitor_directory(directory_path, number_of_scans, scan_interval)

Parameters: directory_path (str), number_of_scans (int), scan_interval (int)

Returns: dict: Complete monitoring results

Description: Main monitoring function that orchestrates scanning and detection

10.2 C Functions

Collections* Initialize()

Returns: Pointer to Collections structure

Description: Initializes collection with capacity of 100 records

void Scan_Directory(Collections *collection, const char *path)

Parameters: collection: Collection to populate, path: Directory to scan

Returns: void

Description: Scans directory and populates collection with file metadata

void CleanUp(Collections **collection)

Parameters: collection: Pointer to collection pointer

Returns: void

Description: Frees all memory allocated for collection and records

11. Best Practices

11.1 Monitoring Strategy

- Start with shorter intervals (30-60 seconds) for critical directories
- Use longer intervals (300+ seconds) for large directories to reduce overhead
- Monitor multiple scan cycles to establish baseline behavior
- Enable email reports for overnight or long-term monitoring
- Exclude temporary directories to reduce noise

11.2 Security Recommendations

- Run with least privileges necessary
- Monitor system directories separately from user directories
- Set up email alerts for critical paths
- Review anomaly thresholds based on environment
- Keep logs of monitoring sessions for audit trails
- Test configuration on non-production systems first

11.3 Performance Optimization

- Compile C library with optimization flags (-O2 or -O3)
- Limit snapshot history to prevent memory growth
- Avoid monitoring network drives with high latency
- Use SSD storage for monitoring system files
- Schedule intensive scans during off-peak hours

Appendix A: File Format Specifications

A.1 Record Structure

Field	Type	Description
file_name	char*	Name of the file
full_path	char*	Complete path to file
file_size	long long	Size in bytes
creation_time	long long	Windows FILETIME
last_access_time	long long	Windows FILETIME
modified_time	long long	Windows FILETIME

Appendix B: Error Codes

Common error codes and their meanings:

Error	Meaning
INVALID_HANDLE_VALUE	Directory cannot be opened
ERROR_ACCESS_DENIED	Insufficient permissions
ERROR_PATH_NOT_FOUND	Path does not exist
ERROR_NOT_ENOUGH_MEMORY	Memory allocation failed
SOCKET_ERROR	Email SMTP connection failed