Universidad de Puerto Rico

Recinto de Bayamón

Departamento de Ciencias en Computadoras

Python Host Based Intrusion Detection System

Jean L. González Milán

SICI-4038

Dr. Nelliud Torres Batista

**Table of Contents**

1. **System Overview…………………………………………………………Page 4**
   1. Key Features
2. **Architecture……………………………………………………………….Page 4**
   1. Main Controller
   2. Monitoring Subsystems
   3. Web Interface
   4. Database
3. **Installation Guide………………………………………………………..Page 4**
   1. Prerequisites
   2. Installation Steps
   3. Running the System
4. **Configuration………………………………………………………………Page 5**
   1. File Monitoring
   2. Process Monitoring
   3. Web Interface
5. **User Guide………………………………………………………………….Page 6**
   1. Logging In
   2. Dashboard Tabs
6. **Monitoring Features……………………………………………………..Page 7**
   1. File Integrity Monitoring
   2. Process Monitoring
   3. Network Monitoring
   4. System Resources
7. **Database Schema………………………………………………………..Page 7**
   1. User Table
   2. Files Table
   3. Events Table
   4. Executables Table
   5. Process Table
   6. ResourceUsage Table
   7. NetworkTraffic Table
8. **Security Considerations……………………………………………….Page 9**
   1. Best Practices
   2. Risk Mitigation
9. **Troubleshooting…………………………………………………………Page 10**
   1. Common Issues
10. **Maintenance………………………………………………………….Page 10**
    1. Database
    2. Configs
11. **Code Explanations………………………………………………….Page 10**
    1. File Monitor (file\_monitor.py)
    2. Network Monitor (network\_monitor.py)
    3. Process Monitor (process\_monitor.py)
    4. Main Controller (main.py)
    5. Web Interface (flask\_app.py)
12. **System Overview**
    * **A Host-Based Intrusion Detection System (HIDS) is a comprehensive security monitoring solution that provides real-time monitoring of system resources, file integrity, network traffic, and process activities. The system consists of:**
      + A backend monitoring subsystem that is Python-based
      + A web-based dashboard interface using the Flask micro web framework
      + An SQLite database for logging all event alerts.
    * **Key features:**
      + Continuous monitoring of system resources (CPU, memory, network)
      + File integrity monitoring with checksum verification
      + Process monitoring with resource usage alerts
      + Network traffic analysis with blacklist filtering
      + User authentication and role-based access
13. **Architecture**  
    **The system follows a modular architecture with different components and approaches:**
    * **Main Controller (main.py)**
      + Coordinates all subsystems
      + Manages parallel execution of components
    * **Monitoring Subsystems**
      + File Monitor (file\_monitor.py)
      + Process Monitor (process\_monitor.py)
      + Network Monitor (network\_monitor.py)
    * **Web Interface**
      + Flask backend (flask\_app.py)
      + HTML frontend (templates/)
      + CSS (static/css)
      + JavaScript (static/js)
    * **Database**
      + SQLite database (hids\_database.db)
      + Multiple tables for different monitoring aspects
14. **Installation Guide**
    * **Prerequisites**
      + Python 3.12+ or higher version
      + Required Python packages:  
         **pip install psutil Flask scapy watchdog sqlite3**
    * **Installation Steps**
      1. Clone or download the HIDS repository
      2. Initialize the database by running the database schema script
      3. Create necessary configuration files:
         + **blacklist.txt** for network monitoring
         + **ignore\_process\_list.txt** for process monitoring
      4. Configure directories to monitor in **main.py**
    * **Running the system**
      + Execute the main controller:

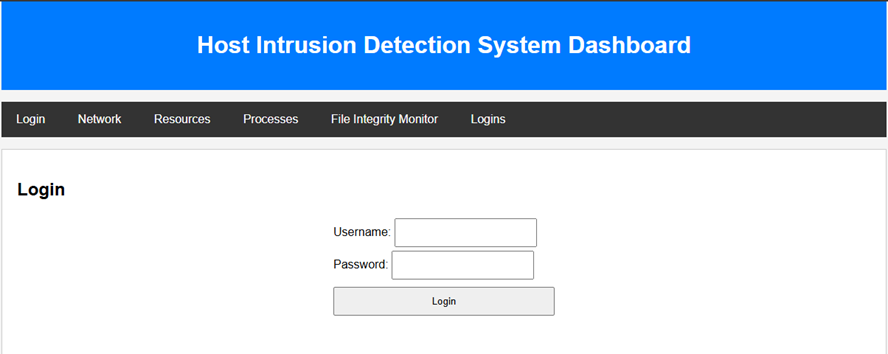
**python main.py**

* + - The web interface will be available:  
       **http://localhost:5000**

1. **Configuration**
   * **File Monitoring**
     1. Edit **main.py** to add directories  
        For Example: **directories\_to\_monitor = ["C:\\Windows\\System32", "/var/log"]**
   * **Process Monitoring**
     1. Exclude safe processes in **ignore\_process\_list.txt**:  
        For example:  
         svchost.exe

Systemd

* + **Web Inteface**
    1. Add user in **flask\_app.py:  
       cursor.execute("INSERT INTO User (username, password\_hash) VALUES (?, ?)",("admin", hash\_password("P@ssw0rd")))**

1. **User guide** 
   * **Loggin in**
2. Open [**http://localhost:5000**](http://localhost:5000)
3. Enter credentials (default: user: admin / password: password  
     
   ***Figure 1:*** *Login Screen*  
   * **Dashboard Tabs**

* Network: Displays suspicious traffic from blacklisted Ips
* Resources: Shows CPU, memory, and network stats
* Processes: Lists running processes with resource usage
* Files: Tracks file changes (creation/modification/deletion)  
  A screenshot of a computer

  AI-generated content may be incorrect.  
  ***Figure 2:*** *View of system menu tabs*

1. **Monitoring Features**
   * **File Integrity** Monitoring
     + Uses CRC32 checksums to detect tampering.
     + Logs:
       - File creations, modifications, deletions
       - User & timestamp
   * **Process Monitoring**
     1. Tracks:
        + CPU usage (%)
        + RAM usage (MB)
        + Process owner
   * **Network Monitoring**
     1. Alerts on:
        + Source/Destination IPs and ports
        + Protocol (TCP/UPD)
        + Packet size and timestamp
   * **System Resources**
     1. Monitors:
        + CPU usage
        + Memory (Total / Used / Available in GB)
        + Network interfaces status
2. **Database Schema**
   * **User table:** Stores system user and authentication details
     1. **Columns:**
        + **user\_id:** Auto-incrementing primary key
        + **username:** Unique login identifier
        + **password\_hash:** SHA-256 hashed password
        + **role:** User permissions level (admin or user
   * **Files Table:** Tracks file system changes
     1. **Columns:**
        + **file\_id:** Unique identifier
        + **filename:** Name of the file
        + **file\_path:** Full absolute path
        + **creation\_time/modification\_time/deletion\_time:** timestamps
        + **hash\_value:** CRC32 checksum of file contents
        + **user:** OS user who modified the file
        + **role:** User privilege level
   * **Events table:** Logs file-related events
     1. **Columns:**
        + **event\_id:** Auto-incrementing key
        + **event\_type:** Action type (CREATED, MODIFIED, DELETED)
        + **event\_time:** When event occurred
        + **file\_id:** Links to Files table
   * **Executables Table:** Catalog of known executables
     1. **Columns:**
        + **executable\_id:** Unique identifier
        + **executable\_name:** Process name (e.g., python.exe)
        + **executable\_path:** Full path to executable
   * **Process Table:**  Tracks running process
     1. **Columns:**
        + **process\_id:** Unique process identifier
        + **executable\_id:** Links to Executables table
        + **start\_time/end\_time:** Process lifecycle timestamps
        + **status:** Current state (running, terminated)
        + **user:** Process owner
        + **role:** Privilege level
   * **ResourceUsage Table:** Records process resource consumption
     1. **Columns:**
        + **cpu\_usage:** Percentage (0-100)
        + **ram\_usage:** Megabytes used
        + **thread\_count:** Number of threads
        + **usage\_level:** Alert classification (NORMAL, LIGHT, MODERATE, CRITICAL)
   * **NetworkTraffic Table:** Logs network packet data
     1. **Columns:**
        + **source\_ip/destination\_ip:** IP addresses
        + **source\_port/destination\_port:** Port numbers
        + **protocol:** Transport layer (TCP, UDP, ICMP)
        + **packet\_size:** Bytes transmitted
        + **timestamp:** When packet was observed

A diagram of a computer program

AI-generated content may be incorrect.***Figure 3:*** *Entity relationship diagram for the Database*

1. **Security Considerations**
   * **Best Practices**
     1. Change default credentials
     2. Restrict access to the database
     3. Use HTTPS in production
   * Risk Nitigation:

| **Risk** | **Solution** |
| --- | --- |
| Unauthorized access | Strong passwords, IP whitelisting |
| DB corruption | Regular backups |
| False positives | Tune thresholds |

1. **Troubleshooting**
   * **Common issues**

| **Problem** | **Fix** |
| --- | --- |
| Web UI not loading | Check flask\_app.py is running |
| Missing file events | Verify directory permissions |
| High CPU usage | Adjust **process\_monitor.py** scan interval |

1. **Maintenance**
2. **Database  
    sqlite3 hids\_database.db ".backup backup.db"**
3. **Configs**
   * Backup:

* blacklist.txt
* ignore\_process\_list.txt

1. **Code Explanations**
   * **File monitor (file\_monitor.py)**
2. **Module overview**The file\_monitor.py module provides comprehensive file system monitoring for the Host Intrusion Detection System (HIDS). It tracks file changes in real-time, verifies file integrity using checksums, and logs all events to an SQLite database
3. **Components**
   * **Insert\_file**def insert\_file(cursor, conn, filename, filepath, creation\_time,

modification\_time, deletion\_time, checksum, user, role):

"""Inserts file metadata into Files table"""

cursor.execute("""

INSERT INTO Files(filename, file\_path, creation\_time,

modification\_time, deletion\_time,

hash\_value, user, role)

VALUES(?, ?, ?, ?, ?, ?, ?, ?)

""", (filename, filepath, creation\_time, modification\_time,

deletion\_time, checksum, user, role))

conn.commit()

* + - Purpose:
      * Stores file metadata including:
      * File path and timestamps (creation/modification/deletion)
      * CRC32 checksum for integrity verification
      * User and role context
    - Error Handling:
      * Rolls back transaction on failure
      * Logs errors to console
  + **insert\_event()**def insert\_event(cursor, conn, event\_type, event\_time, file\_id):

"""Logs file events to Events table"""

cursor.execute("""

INSERT INTO Events(event\_type, event\_time, file\_id)

VALUES(?, ?, ?)

""", (event\_type, event\_time, file\_id))

conn.commit()

* + - **Purpose:**Records file system events with:
      * Event type (Creation, Modified, Deleted)
      * Precise timestamp
      * Foreign key linking to the Files table
  + **calculate\_crc32\_checksum()**def calculate\_crc32\_checksum(filepath):

"""Computes CRC32 checksum in 8KB chunks"""

checksum = 0

with open(filepath, 'rb') as file:

while chunk := file.read(8192):

checksum = zlib.crc32(chunk, checksum)

return format(checksum & 0xFFFFFFFF, '08x')

* + - **Key Aspects:**
      * Processes files in 8KB chunks for memory efficiency
      * Returns 8-character hexadecimal string
      * Handles file access errors gracefully
  + **scan\_directory()**  
    def scan\_directory(directories):

"""Recursively scans directories using os.walk"""

for root, \_, files in os.walk(directory):

for file in files:

filepath = os.path.join(root, file)

checksum = calculate\_crc32\_checksum(filepath)

insert\_file(...) # Store initial file record

insert\_event(...) # Log scan event

* + - **Behavior:**
      * Walks directory tree recursively
      * Calculates checksums for all files
      * Logs baseline state to database
  + **FileEventHandler Class**class FileEventHandler(FileSystemEventHandler):

processed\_files = set() # Prevents duplicate events

def on\_created(self, event): ...

def on\_modified(self, event): ...

def on\_moved(self, event): ...

def on\_deleted(self, event): ...

* + - **Event Handling Workflow:**
      1. **Creation**
         * Generates checksum
         * Logs file metadata + "Creation" event
      2. **Modification**
         * Recomputes checksum
         * Updates file record + logs "Modified" event
      3. **Move/Rename**
         * Tracks old and new paths
         * Preserves original checksum
      4. **Deletion**
         * Marks file as deleted in database
         * Logs "Deletion" event
      5. **Optimizations**
      * Skips directory events
      * Prevents duplicate processing with processed\_files set
  + **monitor\_directories()**

**def monitor\_directories(directories):**

**"""Main monitoring controller"""**

**scan\_directory(directories) # Initial scan**

**observer = Observer()**

**for directory in directories:**

**observer.schedule(FileEventHandler(), directory, recursive=True)**

**observer.start() # Runs until Ctrl+C**

* + **Execution Flow:**
    - Performs initial directory scan
    - Configures Watchdog observer for real-time monitoring
    - Runs indefinitely until keyboard interrupt
  + **Network Monito(network\_monitor.py)**
    1. **Module Overview**

The network\_monitor.py module provides real-time network traffic analysis for the Host Intrusion Detection System (HIDS). It detects suspicious network activity by monitoring communications with blacklisted IP addresses and logs detailed packet information to a SQLite database.

* + 1. **Core Components**
       - **insert\_file()**

def insert\_file(cursor, conn, src\_ip, dst\_ip, src\_port, dst\_port,

protocol, packet\_size, timestamp):

"""Logs network traffic to database"""

cursor.execute("""

INSERT INTO NetworkTraffic(

source\_ip, destination\_ip, source\_port,

destination\_port, protocol, packet\_size, timestamp

) VALUES(?, ?, ?, ?, ?, ?, ?)

""", (src\_ip, dst\_ip, src\_port, dst\_port,

protocol, packet\_size, timestamp))

conn.commit()

* + - * + **Purpose:**

**Stores network packet metadata including:**

**Source/destination IPs and ports**

**Protocol type (TCP/UDP)**

**Packet size and timestamp**

* + - * + **Error Handling:**

**Automatic transaction rollback on failure**

**Error messages logged to console**

* + - * **load\_blacklist()**

def load\_blacklist():

"""Loads IP blacklist from file"""

with open(BLACKLIST\_FILE, "r") as file:

return {

line.strip()

for line in file

if line.strip() and not line.startswith("#")

}

* + - * + **Behavior:**

Skips comments (lines starting with #)

Returns empty set if file not found

Prints loaded IP count for verification

* + - * **log\_packet()**

def log\_packet(packet, blacklist):

"""Analyzes and logs suspicious packets"""

if packet.haslayer(IP):

src\_ip = packet[IP].src

if src\_ip in blacklist:

# Extract protocol data

protocol = "TCP" if packet.haslayer(TCP) else "UDP" if packet.haslayer(UDP) else "OTHER"

# Log to database and console

* + - * + **Packet Analysis Logic:**

Filters for IP-layer packets only

Checks source IP against blacklist

Extracts:

Port numbers (for TCP/UDP)

Protocol type

Packet size

* + - * + Generates timestamp in ISO format
      * **start\_packet\_sniffer()**

def start\_packet\_sniffer():

"""Main monitoring controller"""

blacklist = load\_blacklist()

sniff(

prn=lambda pkt: log\_packet(pkt, blacklist),

store=False # Minimal memory usage

)

* + - * **Configuration:**
        + store=False: Prevents packet accumulation in memory
        + Processes packets in real-time via callback
      * **Operational Flow:**
        + Loads blacklist at startup
        + Initiates promiscuous network sniffing
        + Runs continuously until Ctrl+C
  + **Process Monitor(process\_monitor.py)**
    1. **Module Overview**

The process\_monitor.py module provides comprehensive system process monitoring for the Host Intrusion Detection System (HIDS). It tracks all running processes, their resource consumption, and generates alerts based on configurable thresholds.

* + 1. **Core Components**
       - **Database Operations**
         * **Process Tracking**

cursor.execute("""

INSERT INTO Processes (

executable\_id, start\_time,

end\_time, status, user, role

) VALUES (?, ?, ?, ?, ?, ?)

""", (executable\_id, start\_time, end\_time,

status, getpass.getuser(), user\_role))

* + - * + **Stores:**

**Process lifecycle timestamps**

**Execution status (running/terminated)**

**User and privilege context**

* + - * **Resource Usage Logging**

cursor.execute("""

INSERT INTO ResourceUsage (

process\_id, cpu\_usage,

ram\_usage, thread\_count, usage\_level

) VALUES (?, ?, ?, ?, ?)

""", (process\_id, cpu\_usage, memory\_usage,

thread\_count, usage\_alert))

* + - * + **Metrics Captured:**

CPU utilization (%)

Memory consumption (%)

Thread count

Alert classification

* + - * **main() Monitoring Loop**

while True:

for pid in psutil.pids():

try:

p = psutil.Process(pid)

with p.oneshot(): # Performance optimization

# Collect process metrics

# Apply alert thresholds

# Store in database

* + - * + **Key Behaviors:**

**Iterates through all running processes**

**Uses oneshot() for efficient metric collection**

**Skips processes in ignore list with low resource usage**

**Handles permission errors gracefully**

* + - * **Alert Classification**

if cpu\_usage > CRITICAL\_THRESHOLD or memory\_usage > CRITICAL\_THRESHOLD:

usage\_alert = "CRITICAL"

elif cpu\_usage > MODERATE\_THRESHOLD:

usage\_alert = "MODERATE"

elif cpu\_usage > LIGHT\_THRESHOLD:

usage\_alert = "LIGHT"

else:

usage\_alert = "NORMAL"

* + **Main Controller (main.py)**
    1. **System Orchestration Hub**
       - **Purpose:** Coordinates all HIDS subsystems as parallel processes for concurrent operation.
       - **Key Functions**

def run\_flask\_app(): # Web Dashboard (5000/tcp)

def run\_process\_monitor(): # Process Monitoring

def run\_file\_monitor(): # File Integrity Monitoring

def run\_network\_monitor(): # Network Traffic Analysis

def run\_network\_test(): # Test Packet Generator

* + - * **Execution Flow**
        + Process Initialization

flask\_process = Process(target=run\_flask\_app)

monitor\_process = Process(target=run\_process\_monitor)

* + - * **Concurrent Startup**

flask\_process.start()

monitor\_process.start()

* + - * **Continuous Operation**

flask\_process.join() # Blocks until termination

* + - * **Configuration**
        + Monitored Directories: Configured via directories\_to\_monitor list
        + Production Notes:
        + Replace test directory with system paths (/etc, /bin, etc.)
        + Disable network\_traffic\_test in production
  + **Web Interface (flask\_app.py)**
    1. **Dashboard Service**
       - **Features:**
         * Real-time system monitoring
         * Authentication-protected access
         * JSON API for frontend data
    2. **Core Endpoints**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /login | POST | User authentication |
| /memory\_stats | GET | RAM utilization metrics |
| /cpu\_stats | GET | Processor load data |
| /file\_monitor | GET | File change events |
| /process\_monitor | GET | Running process details |

* + 1. **Security Implementation**

def hash\_password(password):

"""SHA-256 password hashing"""

return hashlib.sha256(password.encode()).hexdigest()

@app.route('/login', methods=['POST'])

def login():

"""Authentication handler"""

if user['password\_hash'] == hash\_password(password):

return jsonify({"success": True}), 200

* + 1. **Data Handling**

conn = sqlite3.connect('hids\_database.db')

conn.row\_factory = sqlite3.Row # Dict-style results

cursor = conn.cursor()

cursor.execute("SELECT \* FROM FilesView")

return jsonify([dict(row) for row in cursor.fetchall()])