🤖Mist Report Automation Setup Guide

This guide will help you set up automated Mist endpoint reporting with Telegram notifications, change detection, cleanup, and health monitoring.

📋Prerequisites

Required Python Packages

```bash
pip3.13 install requests pandas python-telegram-bot configparser schedule openpyxl cryptography
```

Files Needed

- `mist_endpoint_report.py` (your enhanced main script)
- `mist_automation.py` (the automation wrapper)

Files Created During Setup

- `mist_config.ini` (your Mist API configuration)
- `automation_config.ini` (automation settings)

🔧Setup Steps

1. Create Telegram Bot

1. **Create a new bot:**
   - Message @BotFather on Telegram
   - Send `/newbot`
   - Choose a name and username for your bot
   - Save the **bot token** you receive
2. **Get your Chat ID:**
   - Message @userinfobot on Telegram
   - It will reply with your **chat ID**
   - Or message your new bot, then visit: `https://api.telegram.org/bot<BOT_TOKEN>/getUpdates`

2. Create Mist API Configuration

Before setting up automation, you need to configure your Mist API credentials:

```bash
# Create the Mist API configuration file
python3.13 mist_endpoint_report.py --create-config

# This creates: Resources/mist_config.ini
```

Or create `Resources/mist_config.ini` manually:

```ini
[mist]
api_token = your_mist_api_token_here
org_id = your_organization_id_here
base_url = https://api.mist.com
theme = default
days = 7
```

To get your Mist API credentials:

1. **API Token:**
   - Go to Mist Portal
   - Navigate to **Organization → API Tokens**
   - Click **Create Token**
   - Give it a name (e.g., "Endpoint Reports")
   - **Copy the token** (you won't see it again!)
2. **Organization ID:**
   - In your Mist portal, look at the URL
   - Find the part after `org_id=`
   - Example: `https://manage.mist.com/admin/?org_id=12345678-1234-1234-1234-123456789abc`
   - Your org_id is: `12345678-1234-1234-1234-123456789abc`
3. **Base URL (region-specific):**
   - **US (default):** `https://api.mist.com`
   - **EU:** `https://api.eu.mist.com`
   - **APAC:** `https://api.ac2.mist.com`
   - **Other regions:** Check Mist documentation

Test your Mist configuration:

```bash
# Test a basic report to verify credentials
python3.13 mist_endpoint_report.py --config Resources/mist_config.ini --format html --theme default
```

🔒 **Optional: Encrypt your configuration for security:**

```bash
# Method 1: Password-based encryption (prompted for password)
python3.13 mist_endpoint_report.py --encrypt-config

# Method 2: Key-file encryption (more secure for automation)
python3.13 mist_automation.py --create-key
python3.13 config_encryption.py --encrypt Resources/mist_config.ini --key-file Resources/encryption.key
```

3. Create Automation Configuration

```bash
# Create sample automation configuration file
python3.13 mist_automation.py --setup

# This creates: Resources/automation_config.ini
```

4. Edit Automation Configuration

Edit `automation_config.ini` with your actual values:

```ini
[telegram]
bot_token = 1234567890:ABCdefGhIjKlMnOpQrStUvWxYz
chat_id = 123456789
send_success_reports = true
send_error_alerts = true
send_change_alerts = true

[mist]
script_path = ./mist_endpoint_report.py
config_path = ./mist_config.ini
output_formats = html,json
theme = default

[reports]
directory = Reports
keep_days = 30

# ... (rest of config)
```

5. Test Integration

```bash
# Test Telegram connection
python3.13 mist_automation.py --test-telegram

# Should send a test message to your Telegram chat
```

6. Test Single Run

```bash
# Run a single automated report
python3.13 mist_automation.py --run

# This will:
# - Generate a Mist report
# - Compare with previous run
# - Send Telegram notification
# - Store results in database
# - Clean up old files
```

🔒 Configuration Encryption

Why Encrypt Configuration Files?

Your configuration files contain sensitive information:

- **Mist API tokens** - Access to your organization's data
- **Telegram bot tokens** - Control of your notification bot
- **Organization IDs** - Internal identifiers

Encryption Methods

Method 1: Password-Based Encryption

```bash
# Encrypt configuration files (you'll be prompted for password)
python3.13 mist_automation.py --encrypt-configs

# Decrypt for editing
python3.13 mist_automation.py --decrypt-configs
# Edit files...
# Re-encrypt after editing
python3.13 mist_automation.py --encrypt-configs
```

Method 2: Key-File Encryption (Recommended for Automation)

```bash
# Create encryption key file
python3.13 mist_automation.py --create-key

# Encrypt using key file
python3.13 config_encryption.py --encrypt Resources/mist_config.ini --key-file Resources/encryption.key
python3.13 config_encryption.py --encrypt Resources/automation_config.ini --key-file Resources/encryption.key

# The automation will automatically use the key file if present
```

Environment Variable Support

```bash
# Set password via environment variable (for password-based encryption)
export MIST_CONFIG_PASSWORD="your_secure_password"
python3.13 mist_automation.py --run
```

Security Best Practices

1. **Use key-file encryption** for automated systems
2. **Store key files separately** from config files (different servers/locations)
3. **Set restrictive permissions:**

```bash
chmod 600 Resources/encryption.key
chmod 600 Resources/*.ini.enc
```

4. **Don't commit keys to version control** - add to `.gitignore`:

```
Resources/encryption.key
Resources/*.ini
Resources/*.ini.enc
```

5. **Regularly rotate encryption keys** and API tokens

🚀Usage Options

Manual Execution

```bash
# Single run with full automation
python3.13 mist_automation.py --run

# Manual cleanup
python3.13 mist_automation.py --cleanup

# Check system health
python3.13 mist_automation.py --health
```

Scheduled Automation (Daemon)

```bash
# Start the scheduler (runs until stopped)
python3.13 mist_automation.py --schedule

# This will run:
# - Daily reports at configured time
# - Daily cleanup at configured time
# - Weekly health summaries
```

System Service (Linux)

Create `/etc/systemd/system/mist-automation.service`:

```ini
[Unit]
Description=Mist Endpoint Report Automation
After=network.target

[Service]
Type=simple
User=your_username
WorkingDirectory=/path/to/your/scripts
ExecStart=/usr/bin/python3.13 /path/to/mist_automation.py --schedule
Restart=always
RestartSec=30

[Install]
WantedBy=multi-user.target
```

Then:

```bash
sudo systemctl enable mist-automation.service
sudo systemctl start mist-automation.service
sudo systemctl status mist-automation.service
```

Cron Job (Alternative)

```bash
# Edit crontab
crontab -e

# Add daily report at 6 AM
0 6 * * * cd /path/to/scripts && python3.13 mist_automation.py --run

# Add cleanup at 2 AM
0 2 * * * cd /path/to/scripts && python3.13 mist_automation.py --cleanup
```

📊Features Overview

🔔Telegram Notifications

Success Reports Include:

- ⏰Generation time and duration
- 📈Device statistics (total, active, compliance)
- 🔌Connection type breakdown
- 🔄Changes from previous report
- 📄Report file attachment (if enabled)

Error Alerts Include:

- 🚨Failure notification
- ⏰Timestamp and duration
- ❌Error details
- 🔧Action required message

Health Summaries Include:

- 📊24-hour performance stats
- ✅Success rate percentage
- ⚡Average execution time
- 🏥System health status

📈Change Detection

The system tracks:

- **Device count changes** (new/removed devices)
- **Compliance rate changes** (>5% threshold)
- **Activity changes** (devices coming online/offline)
- **Connection type changes** (wireless/wired shifts)

🧹Automated Cleanup

File Cleanup:

- Removes old report files (configurable retention)
- Keeps directory organized
- Logs cleanup activity

Database Cleanup:

- Purges old health logs
- Maintains historical trends
- Optimizes database size

🏥Health Monitoring

Tracks:

- ✅Report success/failure rates
- ⏱️Execution duration trends
- 📊API performance metrics
- 🔍Error patterns

Alerts on:

- Report generation failures
- Slow execution times
- API connectivity issues
- System degradation

📁File Structure

```
your-project/
├── mist_endpoint_report.py      # Main enhanced script
├── mist_automation.py           # Automation wrapper
├── mist_config.ini              # Mist API credentials
├── automation_config.ini        # Automation settings
├── mist_history.db              # SQLite tracking database
├── mist_automation.log          # Log file
└── Reports/                     # Generated reports
    ├── mist_endpoint_report_20241215_060001.html
    ├── mist_endpoint_report_20241215_060001.json
    └── ...
```

🔍Troubleshooting

Common Issues

Telegram not working:

```bash
# Check configuration
python3.13 mist_automation.py --test-telegram

# Verify bot token and chat ID
# Ensure bot can send messages to your chat
```

Mist API credentials issues:

```bash
# Test your Mist configuration first
python3.13 mist_endpoint_report.py --config Resources/mist_config.ini --format html

# Check API token and org_id in Resources/mist_config.ini
# Verify you're using the correct API endpoint for your region
```

Report generation fails:

Verify script paths in automation_config.ini

Check log file: mist_automation.log

Scheduling not working:
```bash
# Check scheduler status
python3.13 mist_automation.py --health

# Verify scheduling is enabled in config
# Check system service status (if using systemd)
```

Log Files

Main automation log:

```bash
tail -f Logs/mist_automation.log
```

Check health database:

```bash
python mist_automation.py --health
```

🔐Security Notes

- **Use encryption** for all configuration files containing sensitive data
- **Store encryption keys separately** from encrypted config files
- **Set restrictive permissions:** `chmod 600 Resources/encryption.key Resources/*.ini.enc`
- **Use environment variables** for passwords in production environments
- **Don't commit sensitive files** to version control (use `.gitignore`)
- **Regularly rotate** API tokens and encryption keys
- **Use key-file encryption** for automated/scheduled operations

🎯Advanced Configuration

Custom Thresholds

Edit `automation_config.ini` to adjust:

- Change detection sensitivity
- Health monitoring thresholds
- Cleanup retention periods
- Notification preferences

Multiple Environments

Use different config files:

```bash
python3.13 mist_automation.py --config Resources/production_config.ini --run
python3.13 mist_automation.py --config Resources/staging_config.ini --run
```

Integration with Monitoring

The SQLite database can be queried by external monitoring tools:

```sql
SELECT * FROM health_log WHERE status = 'failure' ORDER BY timestamp DESC;
SELECT AVG(duration_seconds) FROM health_log WHERE timestamp > datetime('now', '-7 days');
```

📞Support

For issues:

1. Check log files first
2. Verify configuration settings
3. Test individual components
4. Review Telegram bot permissions
5. Check network connectivity

Happy automated reporting! 🎉