# **VTPL System Info & Software Monitor**

# 1. Introduction

The VTPL System Info & Software Monitor is a lightweight, OS-level monitoring client designed to detect and report system configuration and software changes in real-time. It enables centralized monitoring and management of client systems through a server interface, allowing for proactive patch management, audit logging, and inventory tracking of digital assets.

## Goals

- Detect software installation and removal events instantly.  
- Collect detailed system configuration data.  
- Ensure secure and efficient communication with a centralized server.  
- Operate silently in the background with minimal system resource usage.

## Key Features

- Real-time monitoring using Windows Registry hooks  
- Full system info snapshot on startup  
- Lightweight and modular design  
- Extensible for patch deployment and updates

## Target Platform

Windows OS (10 and above), with Administrator access for registry access.

## Audience

This document is intended for developers, system integrators, and maintainers of the monitoring system.

# 2. System Architecture

## 2.1 Overview

The VTPL System Monitor is designed to run on Windows OS machines to monitor system configuration and software changes in real time. It is built in **Python 3.10+** and follows a modular architecture. The system performs:

* A one-time system configuration collection at the time of installation
* Real-time software installation/uninstallation detection changes post installation
* Communication with a central server for reporting and patch management of software installed

## 2.2 Modules involved

|  |  |  |
| --- | --- | --- |
| S. No | Module Name | Description |
| 0 | main.py | Main execution point with either a single info dump with –once or default background + monitoring mode with no args. |
| 1 | system\_info | Collects hardware (OS, CPU, GPU, etc.) and software information via registry using PowerShell commands. |
| 2 | monitor/software\_monitor.py | A background daemon that detects real time installs and uninstalls, sending the log details of the software changes. |
| 3 | monitor/hash.py | Provides SHA-256 function to hash ledger local json and detects field changes. |
| 4 | monitor/state\_manager.py | Manages and compares new states of local json that stores the system information (hardware + software + BIOS). |
| 5 | monitor/startup\_checker.py | Runs at startup each time to compare current system info with previous json snapshot. |
| 6 | api/sender.py | Sends structured data to backend API |
| 7 | constants.py | Defines common constants throughout the project like registry keys, api url, state file, etc |

## Data Flow Summary

1. On first run or using --once:
   * System info is collected and saved to data.json.
2. On system startup (default behavior):
   * The startup checker compares the current system state with data.json (previous system state, if exists).
   * Real-time monitor spawns background threads that watch registry keys for install/uninstall events.
3. When changes are detected:
   * Details are extracted using registry queries.
   * The event is sent to a central API via a POST request.

# 3. Environment Setup & Installation

## 3.1 System Requirements

|  |  |
| --- | --- |
| **Requirement** | **Details** |
| Operating System | Windows 10 or later |
| Python Version | 3.10.x (64-bit) |
| Disk Space | ~30MB for installation |
| Permissions | Admin rights (for registry access + startup) |

## 3.2 Installation Steps (Developer Machine)

1. **Install Python 3.10+**
   * Download from: <https://www.python.org/downloads/release>
   * Ensure Add Python to PATH is selected during setup.
2. **Clone the Repository**
3. git clone [https://github.com/ShubhPundir/System-Info-Collection.git](https://github.com/ShubhPundir/System-Info-Collection)
4. cd system-info-monitor
5. **Install Dependencies**  
   Create a virtual environment and install requirements:
6. python -m venv venv
7. venv\Scripts\activate
8. pip install -r requirements.txt

## 3.3 Required Python Packages

|  |  |
| --- | --- |
| **Package** | **Purpose** |
| psutil | System info (CPU, memory, disks) |
| pywin32 | Windows API access (needed for pythoncom, registry events) |
| requests | HTTP communication with backend API |

These are listed in requirements.txt.

## 3.4 Auto Start on Boot (Client Deployment)

To install the client as a startup service:

1. Add a shortcut of main.py (or a compiled .exe) to:
2. C:\Users\<username>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
3. (Optional) Use pyinstaller to compile to a standalone .exe:
4. pyinstaller --onefile main.py

## 3.5 Server Endpoint Configuration

* The client expects the **company code** and **API URL** to be available (typically passed or configured at build/install time).
* To be entered during installation of app; example:

{

"company\_code": "VTPL123",

"Asset\_Id": "LP014"

}

# 4. Codebase walkthrough

The project follows a modular design where each component is responsible for a specific part of the system monitoring and communication lifecycle.

## 4.1 main.py

* **Purpose**: Main execution point
* **Modes**:
  + --once: Single-time system info dump
  + No args: Runs startup\_checker and real\_time\_install\_monitor
* **Loop**: Keeps client alive in background with daemon threads

## 4.2 system\_info/collector.py

* Collects:
  + CPU model, RAM, storage devices
  + Motherboard and BIOS info
  + Network configuration
  + Installed software list
* Returns structured Python dict
* Uses psutil, platform, subprocess for collection

## 4.3 monitor/startup\_checker.py

* **Purpose**: Runs once on system boot
* **Logic**:
  + Load previous snapshot (data.json)
  + Collect current info
  + Compare via normalized hash (ignores volatile fields)
  + If changed → trigger upload to server
* **Prints diff** for developer visibility

## 4.4 monitor/state\_manager.py

* Reads and writes the system info snapshot from data.json
* Uses standard json module
* Ensures persistence between reboots

## 4.5 monitor/software\_monitor.py

* **Daemon**: Runs in background
* **Uses**: Windows Registry watchers (winreg, win32api)
* **Monitors**:
  + HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
  + HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall
* **On detection**:
  + Extracts new software name/version/publisher
  + Sends structured payload to API
  + Will soon support uninstall detection as well

## 4.6 api/sender.py

* Handles sending data to backend
* Accepts payload dictionary, adds necessary headers
* On 400/401 responses, logs failure
* Future-proofed for retry queue or token auth

# 5. Communication Protocols & API Contracts

This section outlines how the client system interacts with the server backend for data transmission. It includes protocol design, authentication strategy (if any), and expected payload structure.

## 5.1 Overview

* **Communication Protocol**: HTTPS (REST-based)
* **Data Format**: JSON
* **Library Used**: Python requests
* **Error Handling**: HTTP status checks, logs 400/500 responses
* **Authentication**: (Planned or optional) via headers or token (currently not implemented)

## 5.2 System Info Upload (on boot)

* **Triggered by**: startup\_checker.py when a change is detected
* **Endpoint**: POST /api/system-info/upload
* **Headers**

{

“Content-Type”: “application/json”

}

* **Payload Example**
* {
* "companyCode": "main",
* "deviceName": "SHUBH-LAPTOP",
* "assetType": "LAPTOP",
* "uuid": "9DF0AE03-7E94-ED11-8C91-7C5758D7D5B3",
* "macAddress": "7C-57-58-D7-D5-B3",
* "processor": "AMD Ryzen 5 5500U with Radeon Graphics 4.20 GHz",
* "os": "Windows 10 (Build 10.0.26100)",
* "ram": "14278 MB",
* "hddTotal": "0 GB",
* "ssdTotal": "512 GB",
* "serialNumber": "CND3030HJL",
* "manufacturer": "HP",
* "model": "HP 255 G8 Notebook PC",
* "graphicsCard": "AMD Radeon(TM) Graphics",
* "graphicsCardVRam": "2048 MB",
* "antiVirus": "Windows Defender",
* "ipAddress": "192.168.13.40",
* "installedSoftware": [
* {
* "name": "Adobe Genuine Service",
* "version": "9.0.0.29",
* "publisher": "Adobe Inc."
* }, …],
* "bios": [
* {
* "smbiosBiosVersion": "F.32",
* "manufacturer": "Insyde",
* "version": "HPQOEM - 2",
* "releaseDate": "20220802000000.000000+000"
* }
* ]
* }

## 5.3 Real-Time Software Install Alert

* **Triggered by**: monitor/software\_monitor.py (on registry change)
* **Endpoint event**: Install/ Uninstall/ Change
* **Payload Example**
* {
* "company\_code": "VTPL001",
* "machine\_id": "DESKTOP-001",
* "event": "install",
* "timestamp": "2025-07-21 11:20:30",
* "software": {
* "name": "7-Zip 25.00 (x64)",
* "version": "25.00",
* "publisher": "Igor Pavlov"
* }
* }

# 6. Security Considerations

This section highlights the key security practices, known risks, and areas requiring improvement in the current version of the client-server system.

## 6.1 Data Transmission Security

* **Current State**:
  + Data is transmitted over HTTP or HTTPS depending on backend configuration.
  + No encryption at the client level beyond transport-layer (if HTTPS is enabled).
* **Recommended**:
  + Enforce HTTPS by default for all endpoints.
  + Use a valid SSL certificate on the server.

## 6.2 Authentication & Authorization

* **Current State**:
  + The client sends a company\_code as the only form of identification.
  + There is no token-based authentication, user credentials, or signature verification.
* **Risks**:
  + Any actor with the company\_code can POST data to the server.
  + No protection against spoofing or replay attacks.
* **Recommended**:
  + Implement HMAC or JWT-based authentication.
  + Require machine registration using an API key or access token.
  + Rotate keys or tokens periodically.

## 6.3 System Info Sensitivity

* **Data Collected**:
  + System configuration (CPU, RAM, OS, motherboard)
  + Full software inventory with publisher and version
  + Machine hostname
* **Risks**:
  + This data can be sensitive in enterprise environments (e.g., leaking internal tool usage or OS vulnerabilities).
* **Recommended**:
  + Encrypt the payload before transmission.
  + Anonymize hostnames or mask them on the client side.
  + Add an opt-in or redaction config (e.g., hide software from blacklisted vendors).

## 6.4 Local File Security

* **Files**:
  + data.json stores the last known system state (including software inventory).
* **Risks**:
  + Local files are unencrypted and can be modified or spoofed to block monitoring.
* **Recommended**:
  + Sign or hash data.json and verify on load.
  + Store in a hidden or system-protected directory.
  + Use file permissions to restrict tampering.

## 6.5 Monitoring Integrity

* **Current State**:
  + Real-time monitor runs as a user-level script.
  + Can be stopped or killed manually.
* **Recommended Enhancements**:
  + Run the client as a Windows Service with restart policies.
  + Log monitor shutdowns and failures.
  + Add watchdog script or self-healing daemon.

## 6.6 Server-Side Considerations

* **Expected**:
  + Validate all incoming payloads strictly.
  + Sanitize and rate-limit requests to prevent abuse.
  + Log IP addresses and user agents for forensic trails.
  + Implement admin console to audit incoming machine reports.

# 7. Future Enhancements

## 7.1 Functional Requirements

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Two-Way Communication** | Implement full duplex communication between server and client for sending control commands (e.g., remote install, uninstall, or update triggers). |
| **Patch Management** | Allow the server to push software patch actions, making the system act as a centralized package manager for distributed clients. |
| **Delta Reporting** | Instead of sending the full system info, transmit only the changes (diffs) in system configuration, hardware, or software. |
| **Uninstall Monitoring** | Extend current real-time monitor to also detect uninstall events. |
| **GUI Admin Panel** | Build a secure web dashboard to monitor, control, and analyze client device reports in real time. |
| **Cross-Platform Support** | Extend compatibility to Linux/macOS using native mechanisms (e.g., inotify or launchd). |
| **Offline Data Buffering** | If internet is unavailable, cache the updates locally and auto-upload once the connection is restored. |

## 7.2 Security Enhancements

|  |  |
| --- | --- |
| **Area** | **Suggestion** |
| **Authentication** | Use JWT or API keys for secure communication between client and server. |
| **Encryption** | Encrypt local files like data.json and use HTTPS/TLS for all client-server communication. |
| **Integrity Check** | Add checksum/CRC for transmitted data and verify upon receipt. |
| **Code Signing** | Sign the client binary and installer to prevent tampering or spoofed distribution. |
| **Access Control** | Implement role-based access in the admin dashboard to restrict visibility and control. |

## 7.3 Monitoring

|  |  |
| --- | --- |
| **Component** | **Recommendation** |
| **Central Logging** | Ship logs from all clients to a centralized dashboard. |
| **System Health Pings** | Clients periodically send heartbeat pings to confirm operational status. |
| **Detailed Audit Logs** | Log all actions taken (software installs, config changes, etc.) with user, timestamp, and origin. |

## 7.4 Modularization & Maintenance

|  |  |
| --- | --- |
| **Topic** | **Suggestions** |
| **Plugin Architecture** | Break features into pluggable modules (e.g., software monitoring, hardware, services, etc.). |
| **Background Service Mode** | Convert client to a proper Windows service or systemd unit to survive reboots and run silently. |
| **Installer & Updater** | Create an automatic installer and self-updater for the client with version control. |
| **Better Config System** | Use config.ini or YAML files to allow flexible configuration of polling intervals, endpoints, and logging behavior. |