**🛡️ GPO Backup and Reapply Tool**

**Secure Local Group Policy Backup & Restoration Utility**

**📘 1. Overview**

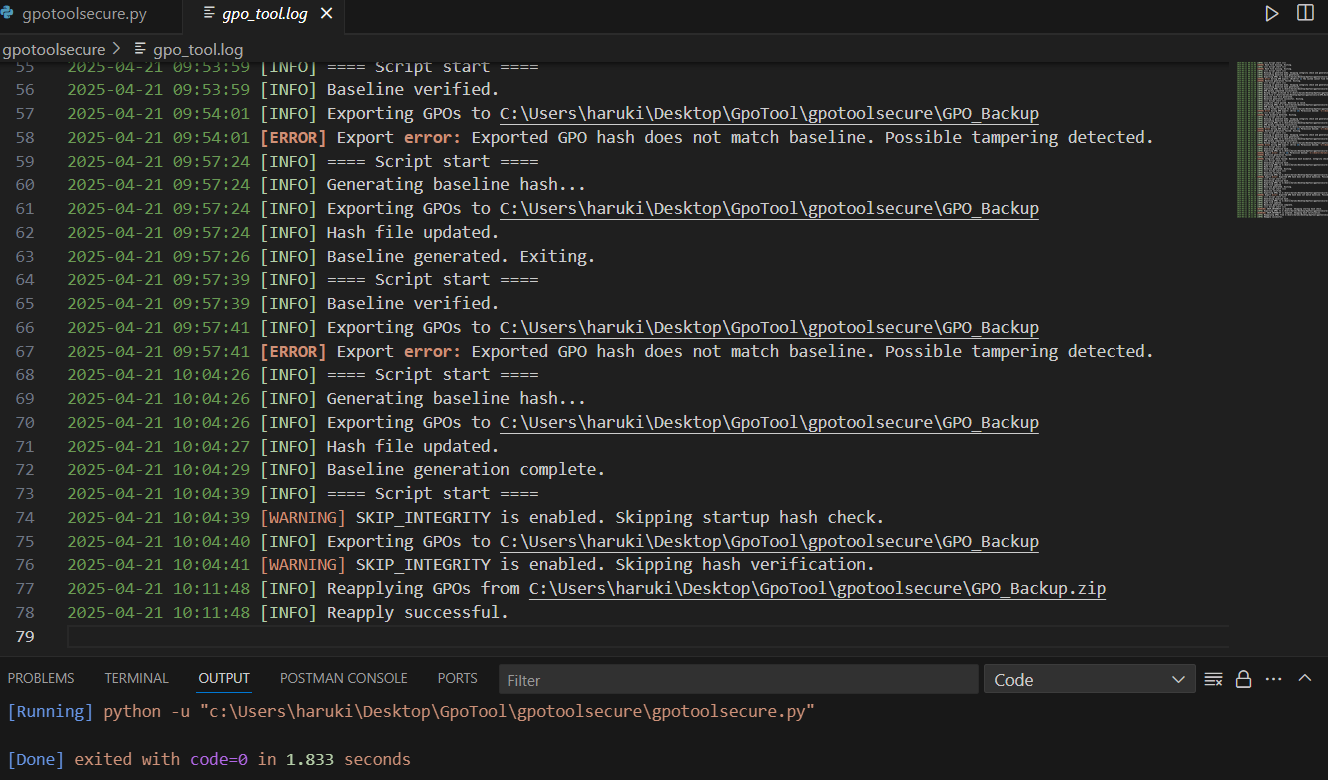
This tool is a secure and user-friendly Python-based application designed to **backup** and **reapply** Windows **Local Group Policy Objects (GPOs)**. It is especially useful in environments where consistent GPO settings are essential for compliance, security, or rapid rollback — such as on isolated workstations which can be scheduled on task scheduler.

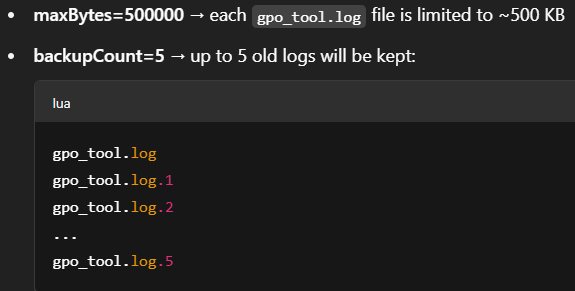
The tool features:

* Reliable GPO exporting using LGPO.exe

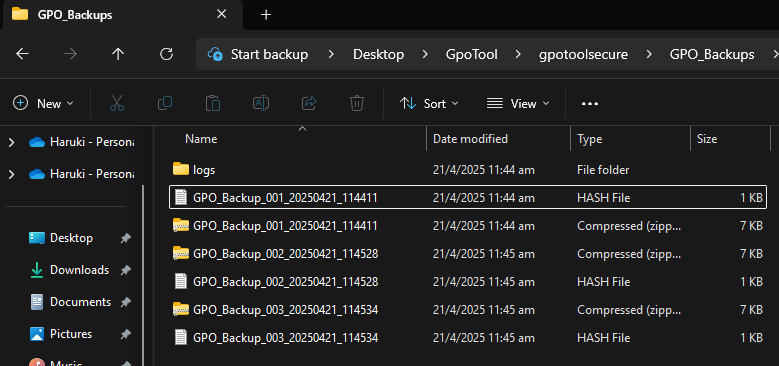
Link for reference: <https://www.reddit.com/r/msp/comments/m215uh/local_group_policies_using_lgpo_lgpoexe_to_deploy/?rdt=40289>

* Detailed logging and monitoring for audit and troubleshooting utilizing **RFS(Rotating File Stream)** for lightweight feasibility



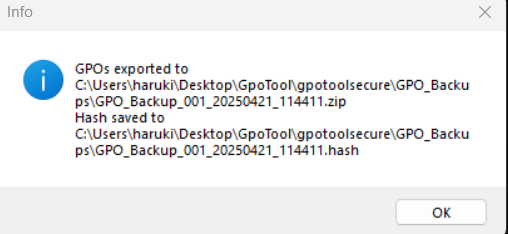


* Numbered and timestamped backups detailed naming convention

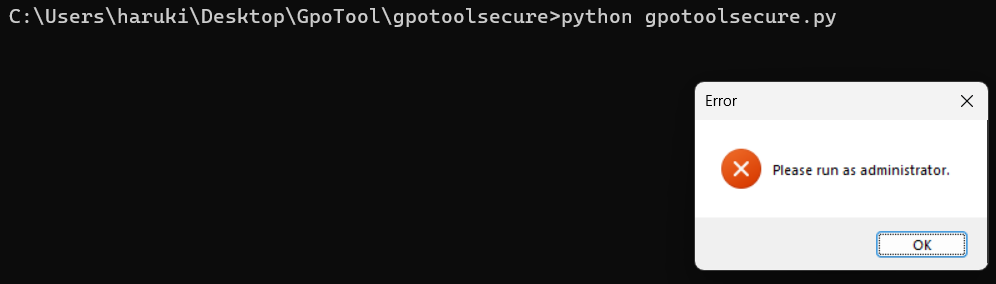


* Hash-based integrity verification using **SHA-256** hashing algorithm

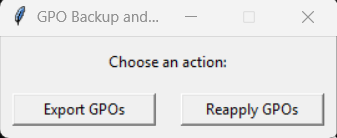




* Admin-only access enforcement



* Intuitive GUI for exporting or reapplying policies

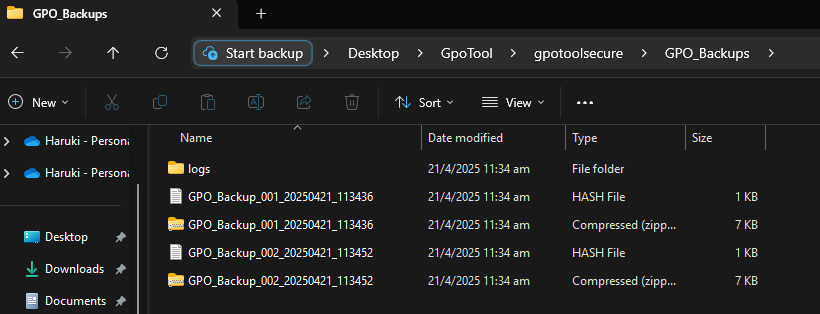


**🔧 2. Features and Functional Flow**

**2.1 Export GPO**

When the user clicks **“Export GPOs”**:

* The tool runs lgpo.exe /b to export current local GPOs to a temporary folder.

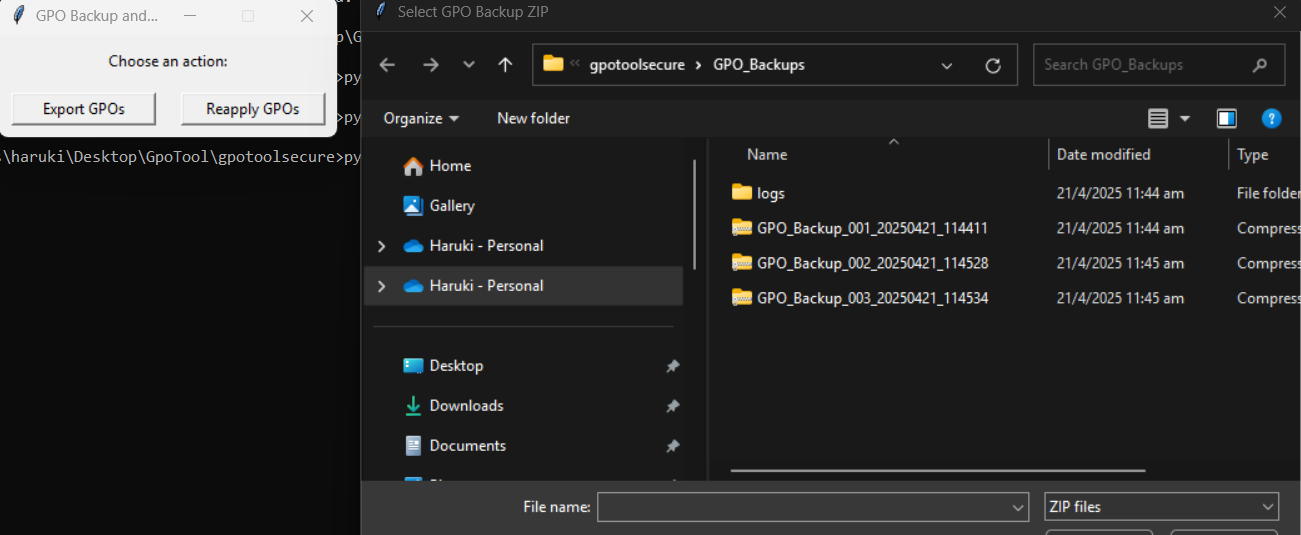


* It zips the export into a file like:  
  **GPO\_Backup\_001\_20250421\_1800.zip**
* It computes a **SHA-256 hash** of the .zip file and saves it as:  
  **GPO\_Backup\_001\_20250421\_1800.hash**
* The .zip and .hash files are stored in ./GPO\_Backups
* The hash file is marked **read-only** to prevent tampering.
* A popup notifies the user of success or failure.

**2.2 Reapply GPO**

When the user clicks **“Reapply GPOs”**:

* The tool prompts the user to select a .zip backup file.
* It automatically locates the corresponding .hash file.



* It verifies the .zip file’s integrity by comparing its computed hash to the stored hash.
  + This prevents unauthorized modification or use of outdated files.
* The zip is extracted to a temporary folder.
* The contents are applied via lgpo.exe /g.
* The temp folder is removed afterward.

🔐 Security Features

**1. Administrator Privilege Enforcement**

**Code:**

python

CopyEdit

if not is\_admin():

show\_error\_popup("Please run as administrator.")

sys.exit(1)

✅ What it does:  
Ensures the script only runs with elevated privileges (admin rights).

🔒 Why it’s important:

* Exporting and applying GPOs affects critical system settings.
* Prevents unauthorized or low-privilege users from tampering with policies.

**2. File Integrity Verification (Hashing)**

**Code**:

python

CopyEdit

def compute\_file\_hash(path):

...

with open(hash\_name, "w") as hf:

hf.write(hash\_val)

✅ What it does:  
Calculates and stores a SHA-256 hash of every GPO backup. On reapply, it verifies the ZIP file against the stored hash.

🔒 Why it’s important:

* Prevents tampered or malicious GPO backups from being applied silently.
* Ensures authenticity and integrity of backed-up policies.
* Detects unauthorized changes to .zip backups.

**3. Optional Integrity Bypass for Development**

**Code:**

python

CopyEdit

SKIP\_INTEGRITY = True

✅ What it does:  
Lets you skip the hash verification (used in development or non-production testing).

🔒 Why it’s important:

* Adds flexibility for developers
* Ensures strict validation in production, but easy testing during development

➡️ Best practice: set SKIP\_INTEGRITY = False in deployment.

**4. Hash File Protection**

**Code:**

python

CopyEdit

subprocess.run(["attrib", "+R", hash\_name])

✅ What it does:  
Makes the .hash file read-only, preventing accidental or malicious edits.

🔒 Why it’s important:

* Prevents tampering with trusted baseline hashes
* Maintains non-repudiation of GPO file integrity

**5. Logging with Rotating File Stream (RFS)**

**Code:**

python

CopyEdit

handler = RotatingFileHandler(log\_file, maxBytes=500000, backupCount=5)

✅ What it does:  
Captures all activity into a rotating log system (gpo\_tool.log), including:

* Success/failure of exports
* Hash mismatches
* File paths used
* Administrative checks

🔒 Why it’s important:

* Creates an audit trail for accountability
* Detects suspicious or repetitive failures
* Supports post-incident investigation

**6. Safe Temporary Extraction**

**Code:**

python

CopyEdit

temp\_extract = os.path.join(BACKUP\_DIR, \"temp\_import\")

...

shutil.rmtree(temp\_extract)

✅ What it does:  
Extracts and deletes temporary GPO content after applying them.

🔒 Why it’s important:

* Prevents GPO residue from being reused or altered
* Reduces attack surface for unauthorized file injection

🛡️ Summary: Why These Security Features Matter

| ***Security Feature*** | ***Risk Mitigated*** | ***Value Added*** |
| --- | --- | --- |
| ***Admin Privileges Required*** | ***Unauthorized execution*** | ***Ensures script control by trusted users*** |
| ***Hash Integrity Check*** | ***Tampered backups*** | ***Assures policy authenticity*** |
| ***Read-only Hash Files*** | ***Accidental/malicious changes*** | ***Preserves trusted baselines*** |
| ***Secure Logging (RFS)*** | ***No visibility / troubleshooting*** | ***Full auditability, easier debugging*** |
| ***Cleanup of Temp Files*** | ***Reuse or manipulation of sensitive files*** | ***Cleaner environment, prevents leakage*** |
| ***Dev-mode Toggle (SKIP\_INTEGRITY)*** | ***Developer roadblocks*** | ***Safer testing without compromising prod*** |