PHASE 5 — SECURITY, AUTH & HUMAN-IN-THE-LOOP CONTROL

“Nothing dangerous runs without you.”

This phase adds:

- Screen-lock detection
- Keyring/password interception
- Secure Telegram password entry
- YES / NO confirmations
- System-level commands (/sysrest, /syslogout)
- Safety rails everywhere

🎯 PHASE 5 OBJECTIVES

✅ Detect screen lock / wake

✅ Detect Antigravity keyring popups

✅ Request password via Telegram

✅ Inject password safely (no logs, no disk)

✅ /sysrest — system restart (guarded)

✅ /syslogout — logout user (guarded)

✅ Confirmation workflow engine

🧠 CORE SECURITY PRINCIPLES (LOCK THESE)

1. Passwords are never stored
2. Passwords live in memory only
3. Passwords are wiped immediately after use
4. All destructive actions require explicit confirmation
5. No AI involvement in password handling

📁 DIRECTORY EXPANSION (PHASE 5)

```
security/
├── detector.py          # Lock & keyring detection
├── password_flow.py     # Secure password handling
├── confirm.py           # YES/NO validation engine
│
agent/
├── sysrest.py
├── syslogout.py
```

1️⃣ SCREEN LOCK & KEYRING DETECTION

Detection Signals

- Window titles:

    - Authentication Required
    - Keyring
    - Unlock
- OCR keywords:
    - password
    - enter password
    - keyring

---

security/detector.py

```
import pyautogui
import pytesseract
import cv2
import numpy as np

KEYWORDS = ["password", "keyring", "authentication"]

def detect_password_prompt():
    screenshot = pyautogui.screenshot()
    img = cv2.cvtColor(np.array(screenshot), cv2.COLOR_RGB2BGR)

    text = pytesseract.image_to_string(img).lower()

    for key in KEYWORDS:
        if key in text:
            return True

    return False
```

This runs:

- After screen wake
- After Antigravity launch
- On UI focus failure

2️⃣ SECURE PASSWORD FLOW

Lifecycle

```
Prompt detected
→ Telegram asks for password
→ User sends password
→ Inject password
→ Clear memory
→ Continue execution
```

---

security/password_flow.py

```
import pyautogui
import time

_temp_password = None

def set_password(pw: str):
    global _temp_password
    _temp_password = pw

def inject_password():
    global _temp_password

    if not _temp_password:
        return False

    pyautogui.typewrite(_temp_password, interval=0.05)
    pyautogui.press("enter")

    # wipe immediately
    _temp_password = None
    return True
```

⚠️ No logging.

⚠️ No file writes.

⚠️ No retries without user.

3️⃣ TELEGRAM PASSWORD REQUEST FLOW

In

telegram/bot.py

(conceptual hook)

```
from security.detector import detect_password_prompt
from security.password_flow import set_password, inject_password
```

When detected:

```
await update.message.reply_text(
    "🔐 Password required.\nSend the system password to continue."
)
```

When user replies:

```
set_password(update.message.text)
inject_password()

await update.message.reply_text("✅ Password accepted. Continuing…")
```

4️⃣ CONFIRMATION ENGINE (YES / NO)

We do not trust accidental taps.

---

security/confirm.py

```
_pending = {}

def request_confirmation(user_id, action):
    _pending[user_id] = action
```

```
    return f"⚠️ Confirm action `{action}`\nReply YES or NO"

def validate_confirmation(user_id, response):
    if user_id not in _pending:
        return None

    if response == "YES":
        return _pending.pop(user_id)
    else:
        _pending.pop(user_id)
        return None
```

5️⃣

/sysrest

— SYSTEM RESTART (GUARDED)

agent/sysrest.py

```
import subprocess
from security.confirm import request_confirmation,
validate_confirmation

async def system_restart(update, text):
    user_id = update.effective_user.id

    if text.strip() == "/sysrest":
        msg = request_confirmation(user_id, "SYSTEM_RESTART")
        await update.message.reply_text(msg)
        return

    action = validate_confirmation(user_id, text.strip())
    if action == "SYSTEM_RESTART":

        await update.message.reply_text("🔄 Restarting system…")
        subprocess.run(["sudo", "reboot"])
```

6️⃣

/syslogout

— USER LOGOUT (GUARDED)

agent/syslogout.py

```
import subprocess
from security.confirm import request_confirmation,
validate_confirmation

async def system_logout(update, text):
    user_id = update.effective_user.id

    if text.strip() == "/syslogout":
        msg = request_confirmation(user_id, "SYSTEM_LOGOUT")
        await update.message.reply_text(msg)
        return

    action = validate_confirmation(user_id, text.strip())
    if action == "SYSTEM_LOGOUT":

        await update.message.reply_text("👋 Logging out user…")
        subprocess.run(["gnome-session-quit", "--logout",
"--no-prompt"])
```

7️⃣ ROUTER UPDATE (PHASE 5)

Add:

```
"/sysrest": system_restart,
"/syslogout": system_logout,
```

🧪 SECURITY TEST MATRIX (MANDATORY)

Scenario	Expected

Screen locks	Telegram asks for password
Wrong password	System rejects, retry
Keyring popup	Password flow triggers
/sysrest	YES required
/syslogout	YES required
Random text	No action

If any auto-executes → bug.

🧠 WHAT YOU HAVE NOW

At this point, CLAWBOLT is:

✅ AI-optional

✅ Crash-resilient

✅ Remote-controlled

✅ Security-aware

✅ Human-validated

This is operator tooling, not a chatbot.