Behavior-Based Malware Detector with Python

Introduction

Instead of scanning files like any ordinary Antivirus program, this program monitors what a program does.

What it detects:

- High CPU consumption by a process
- Suspicious process creation
- File drops in system folders
- Registry modifications
- Network connections to unknown IPs
- High-risk API call patterns

There are limitations as well of this project including:

- False positives on CPU-intensive legit apps
- netstat parsing is OS-dependent
- Not kernel-level monitoring
- Evasion techniques can bypass user-space detection

Future work can include the following aspects:

- ML-based behavior classification
- Process lineage graphing
- Memory inspection
- Integration with Elastic/Splunk
- Real-time Sysmon correlation

Methodology

I will be creating this project with Python with the following libraries and each of their use cases.

- psutil - process & resource monitoring
- watchdog - file system changes
- subprocess - controlled execution
- scapy - network behavior

- yara-python - hybrid rules
- Flask - for the GUI of the application
- And other imports that will support the extras such as datetime, logging, time, and others.

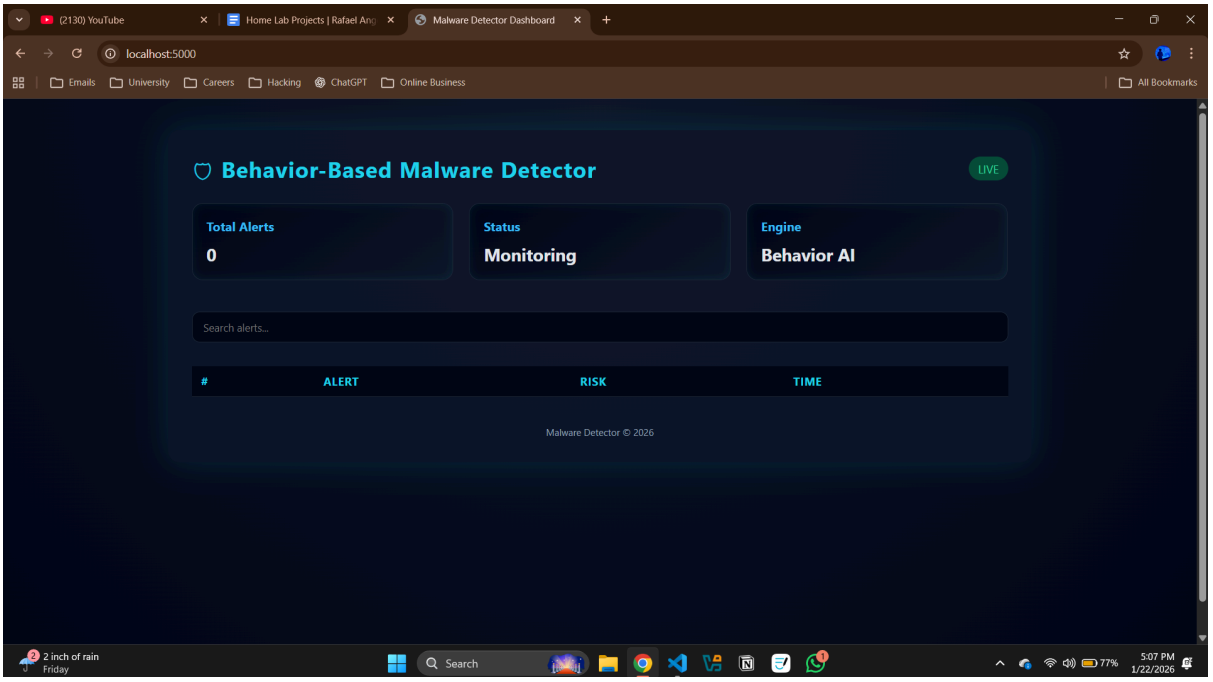
Together with Python, I will also use Sysmon logs. Sysmon (System Monitor) is a free Microsoft Sysinternals tool that acts as a Windows system service, providing deep, detailed logging of system activity, far beyond standard logs, for security and threat hunting

Results

The code can be found at the GitHub repository below:
https://github.com/RafaelAngeloChristianto/behavior-based-malware-detector
The GitHub repository is also completed with the DISCLAIMER and LICENSE text files and also the setup guide, together with the code.

This program is designed to detect these following factors mentioned below:

- High CPU usage
- Suspicious processes
- Writing to startup folders
- Execute from Appdata/temp folders
- PowerShell abuse pattern



By default after running, on a completely safe and clean computer, it will not return anything, because it will only show malware behavior if there are any malware in the computer.

In the terminal, we can see HTTP GET requests to /api/alerts and it returned code 200, which means it was successful.

```
127.0.0.1 - - [22/Jan/2026 17:09:32] "GET /api/alerts HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [22/Jan/2026 17:09:32] "GET /api/alerts HTTP/1.1" 200 -
127.0.0.1 - - [22/Jan/2026 17:09:36] "GET /api/alerts HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [22/Jan/2026 17:09:36] "GET /api/alerts HTTP/1.1" 200 -
127.0.0.1 - - [22/Jan/2026 17:10:27] "GET /api/alerts HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [22/Jan/2026 17:10:27] "GET /api/alerts HTTP/1.1" 200 -
```

It keeps on repeating log entries because the web dashboard keeps on requesting new data from the server regularly. It keeps on sending HTTP GET requests to /api/alerts endpoint to get the most recent alerts, which is then handled by the Flask server. This is necessary for the application to display real-time alerts.

I also left some processes that may take a lot of CPU processing power, but are totally normal processes, if they are performing a heavy task and taking a lot of CPU power the program may flag it as suspicious even though it is normal.

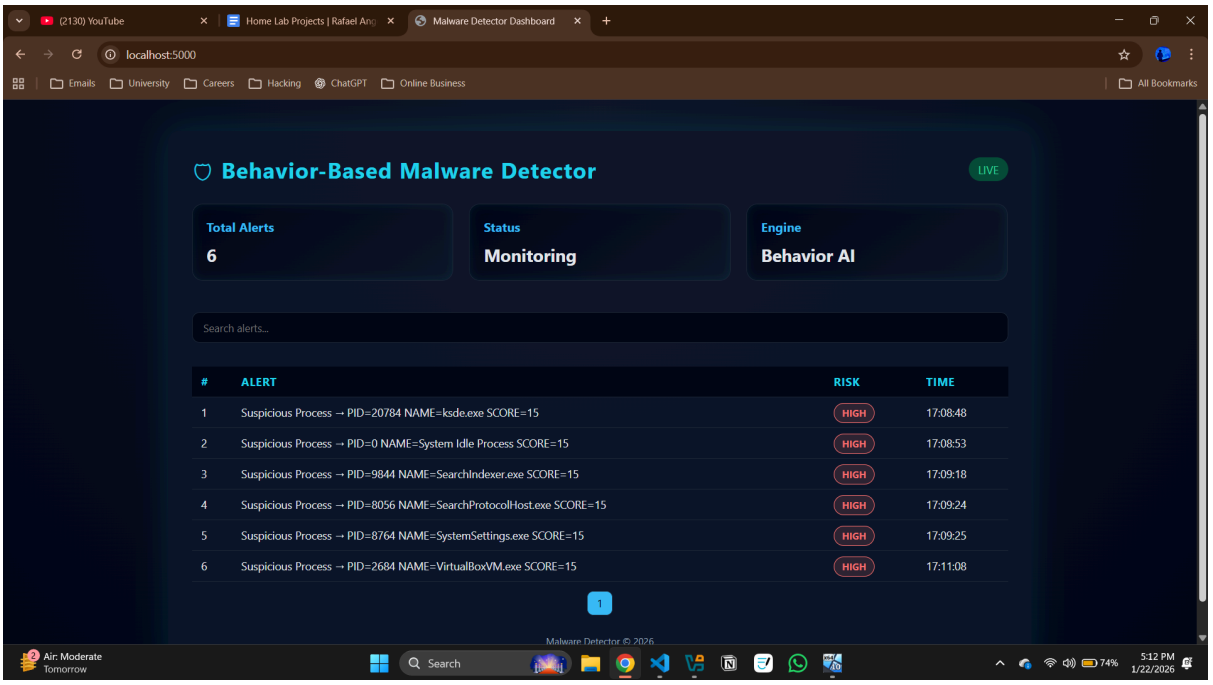
Therefore the processes which are given below are normal system processes. My program stated the CPU process would be flagged if it went over 40. The list of trusted and normal processes is given below:

```
TRUSTED_PROCESSES = {
    "svchost.exe",
    "explorer.exe",
    "onedrive.exe",
    "avp.exe",
    "code.exe",
    "msedge.exe",
    "msedgewebview2.exe",
    "chrome.exe",
    "firefox.exe",
    "python.exe",
    "system",
    "idle",
}
```

The code screenshot below tells us if there are any beaconing or C2 servers interacting with my machine. It uses subprocess to run netstat and -ano to discover which connections are being established right now with their PIDs.

```python
def monitor_network_beaconing():
    result = subprocess.run(["netstat", "-ano"], capture_output=True, text=True, shell=True)
    for line in result.stdout.splitlines():
        if "ESTABLISHED" in line:
            parts = line.split()
            if len(parts) >= 5:
                remote_addr = parts[2]
                pid = parts[-1]
                try:
                    pid = int(pid)
                    proc = psutil.Process(pid)
                    name = proc.name()

                    if is_external_ip(remote_addr):
                        now = time.time()
                        connection_tracker[(pid, remote_addr)].append(now)

                        if len(connection_tracker[(pid, remote_addr)]) >= 4:
                            score_process(pid, name,
                                          f"Beaconing → {remote_addr}", 5)
                except:
                    continue
```

Test Case



I launched VirtualBox, the following exe above may seem normal such as VirtualBox.exe or ksd.exe for Kaspersky, but they are taking a lot of CPU power therefore they are displayed here.

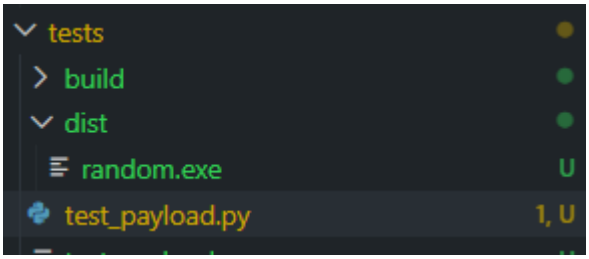
#	ALERT	RISK	TIME
1	Registry Persistence → Opera GX Stable = C:\Users\LENOVO\AppData\Local\Programs\Opera GX\opera.exe	HIGH	11:29:57
2	Registry Persistence → Discord = "C:\Users\LENOVO\AppData\Local\Discord\Update.exe" --processStart Discord.exe	HIGH	11:29:57
3	Registry Persistence → Opera GX Browser Assistant = C:\Users\LENOVO\AppData\Local\Programs\Opera GX\assistant\browser_assistant.exe	HIGH	11:29:57
4	Suspicious Process → PID=2800 NAME=SecurityHealthSystray.exe SCORE=7.8	HIGH	11:30:00
5	Suspicious Process → PID=16512 NAME=RtkAudUService64.exe SCORE=7.8	HIGH	11:30:01
6	Suspicious Process → PID=15184 NAME=FileSyncHelper.exe SCORE=15.8	HIGH	11:30:01
7	Suspicious Process → PID=17204 NAME=OneDrive.Sync.Service.exe SCORE=15.8	HIGH	11:30:01
8	Registry Persistence → Opera GX Stable = C:\Users\LENOVO\AppData\Local\Programs\Opera GX\opera.exe	HIGH	11:30:01
9	Registry Persistence → Discord = "C:\Users\LENOVO\AppData\Local\Discord\Update.exe" --processStart Discord.exe	HIGH	11:30:01
10	Registry Persistence → Opera GX Browser Assistant = C:\Users\LENOVO\AppData\Local\Programs\Opera GX\assistant\browser_assistant.exe	HIGH	11:30:01

1 2

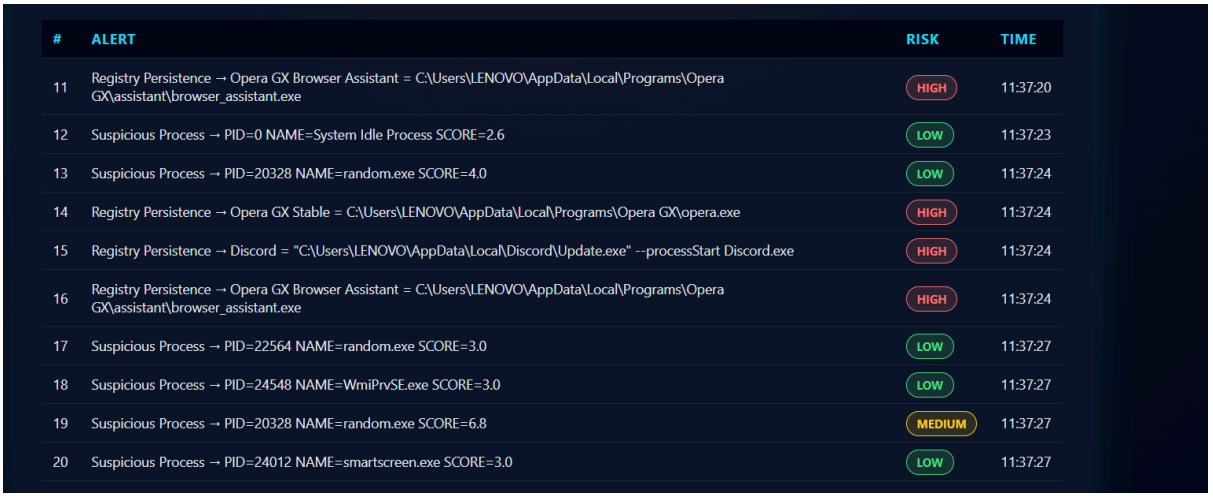
For the screenshot below, I have made a program called test_payload.py that will also be included in the GitHub repository. The program is as follows written in Python:

```python
import time
import requests

print("Test payload running...")

while True:
    try:
        requests.get("http://rafaelangelo.site")
        time.sleep(5)
    except:
        pass
```

So what it does it basically keeps on performing a GET request to my portfolio website to simulate C2 or beaconing behavior. Then I use py installer to make it into an executable under the dist folder.



Then I executed the file and yes it is recorded in the program.



#	ALERT	RISK	TIME
11	Registry Persistence → Opera GX Browser Assistant = C:\Users\LENOVO\AppData\Local\Programs\Opera GX\assistant\browser_assistant.exe	HIGH	11:37:20
12	Suspicious Process → PID=0 NAME=System Idle Process SCORE=2.6	LOW	11:37:23
13	Suspicious Process → PID=20328 NAME=random.exe SCORE=4.0	LOW	11:37:24
14	Registry Persistence → Opera GX Stable = C:\Users\LENOVO\AppData\Local\Programs\Opera GX\opera.exe	HIGH	11:37:24
15	Registry Persistence → Discord = "C:\Users\LENOVO\AppData\Local\Discord\Update.exe" --processStart Discord.exe	HIGH	11:37:24
16	Registry Persistence → Opera GX Browser Assistant = C:\Users\LENOVO\AppData\Local\Programs\Opera GX\assistant\browser_assistant.exe	HIGH	11:37:24
17	Suspicious Process → PID=22564 NAME=random.exe SCORE=3.0	LOW	11:37:27
18	Suspicious Process → PID=24548 NAME=WmiPrvSE.exe SCORE=3.0	LOW	11:37:27
19	Suspicious Process → PID=20328 NAME=random.exe SCORE=6.8	MEDIUM	11:37:27
20	Suspicious Process → PID=24012 NAME=smartscreen.exe SCORE=3.0	LOW	11:37:27

System Components

Here are my states that I used for this program, and what each of their purpose is:

- alerts_feed - Stores all alerts for the Flask dashboard.
- process_scores - Maps PIDs into risk score.
- alerted_events - Prevents duplicate alerts for the same PID and behavior.
- connection_tracker - Tracks timestamps of network connections to detect beaconing.

Starting below are the functions that are used throughout this program:

- push_alert(message, risk=”Medium”)
    - It is the central alerting system that is complete with the time, message, and risk level, it appends it to the alerts_feed array and this is how the program talks to the Flask dashboard.
- score_process(pid, name, reason, points)
    - This is the main process, it ignores trusted and normal system processes so it would not be displayed in the Flask dashboard. It adds the risk points, it also logs the reason, then it checks if the score exceeds the threshold, and then raises an alert if it detects bad behavior. To add, it also does not use a feature where it raises an alert after the first bad thing, it uses cumulative behavior scoring, which goes up to date with the modern standard.
- decay_scores()

    - This one prevents permanent suspicion, so if a process is suspicious before doing something suspicious before, it should not be flagged forever. That can be done by degrading the score.
- monitor_new_process()
    - It detects new processes by comparing the current and known PIDs, and if it is a new PID it creates a new alert. This is necessary since malware likes to spawn as child processes to avoid detection.
- monitor_processes()
    - It detects processes which use a lot of CPU, where this program is set to have a threshold for this to be 40%. This is because malware can spike up CPU usage.
- monitor_parent_child()
    - It detects suspicious parent and child relationship processes. If internet browsers or Office applications spawn child processes, it is most likely malware, because malware likes to use those.
- is_external_ip(ip)
    - This is to check if the device establishes a connection to a public IP which is outside the LAN, which could indicate C2 server and not going to 192.168.x.x.
- monitor_network()
    - This is also to check C2 behavior by checking netstat -ano to get established connections to external IPs and get their PIDs and it flags unknown external connections.
- monitor_network_beaconing()
    - It detects the beaconing malware behavior, it tracks if it makes regular connections to a single IP. If it happens very repeatedly, it gets flagged.
- initial_yara_scan()
    - This looks at the YARA rules and scan processes' binaries and if matched it will flag the process as suspicious. This combines signature and anomaly detection, to make it better.
- monitior_registry()
    - This detects startup persistence. It scans Run and RunOnce keys and if EXE/PS1/BAT/VBS is found, it will get an alert. This is because malware wants to stay in after being rebooted.

- MalwareEventHandler.on_created()
    - This part watches the files using the watchdog library. It detects if there is a new file creation at Windows/System/Public folders, it gives an alert. This is because malware likes to hide their files there.

What happens in __main__:

1. Starts logging
2. Starts filesystem watcher
3. Runs YARA scan
4. Enters monitoring loop:
    - Process creation
    - CPU abuse
    - Parent-child
    - Network
    - Beaconing
    - Registry
    - Score decay
5. Sleeps 3 seconds, then repeat