

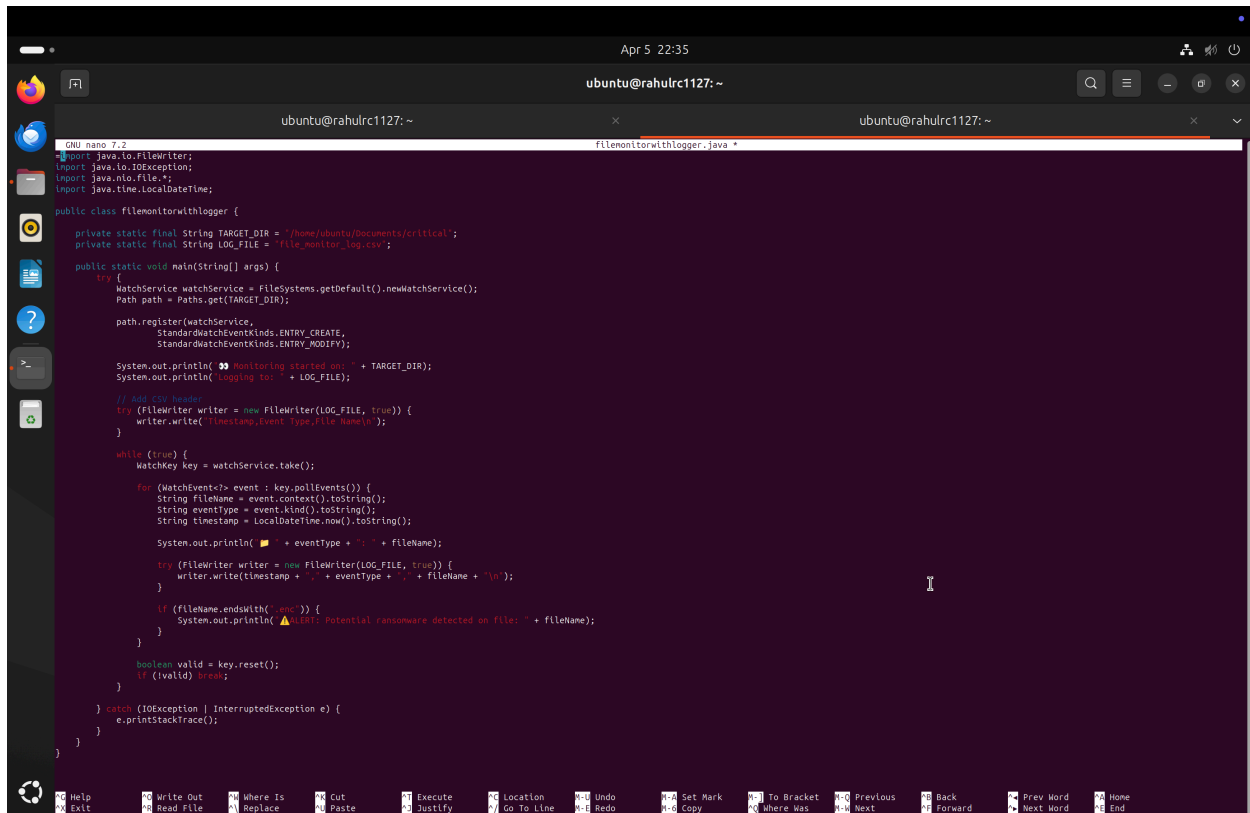
Group 12

Step 4: Monitoring

We'll use:

- **Java**
- **WatchService API** to monitor files in real-time
- **CSV file logging** (easy to implement and counts as structured)

The creation of filemonitorwithlogger.java



The screenshot shows a terminal window with the GNU nano 7.2 editor open. The file being edited is filemonitorwithlogger.java. The code is a Java program that monitors a directory for file changes and logs them to a CSV file. The code includes imports for java.io.FileWriter, java.io.IOException, java.nio.file.Path, and java.time.LocalDateTime. It defines a public class filemonitorwithlogger with a main method. The main method sets up a WatchService, registers it with a path, and enters a loop to monitor for events. It logs the event details to a CSV file and prints them to the console. The code also includes a check for potential ransomware detection based on file extensions.

```
GNU nano 7.2 filemonitorwithlogger.java
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Path;
import java.time.LocalDateTime;

public class filemonitorwithlogger {
    private static final String TARGET_DIR = "/home/ubuntu/documents/critical";
    private static final String LOG_FILE = "file_monitor_log.csv";

    public static void main(String[] args) {
        try {
            WatchService watchService = FileSystems.getDefault().newWatchService();
            Path path = Paths.get(TARGET_DIR);
            path.register(watchService,
                StandardWatchEventKinds.ENTRY_CREATE,
                StandardWatchEventKinds.ENTRY_MODIFY);

            System.out.println("Monitoring started on: " + TARGET_DIR);
            System.out.println("Logging to: " + LOG_FILE);

            // Add CSV header
            try (FileWriter writer = new FileWriter(LOG_FILE, true)) {
                writer.write("timestamp,event_type,filename\n");
            }

            while (true) {
                WatchKey key = watchService.take();

                for (WatchEvent<?> event : key.pollEvents()) {
                    String fileName = event.context().toString();
                    String eventType = event.kind().toString();
                    String timestamp = LocalDateTime.now().toString();

                    System.out.println(timestamp + " " + eventType + " " + fileName);

                    try (FileWriter writer = new FileWriter(LOG_FILE, true)) {
                        writer.write(timestamp + " " + eventType + " " + fileName + "\n");
                    }

                    if (fileName.endsWith(".exe")) {
                        System.out.println("⚠️ ALERT: Potential ransomware detected on file: " + fileName);
                    }
                }

                boolean valid = key.reset();
                if (!valid) break;
            }
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Group 12

This is the output for monitoring started in critical folder.

```

ubuntu@rahulrc1127:~$ nano filemonitorwithlogger.java
ubuntu@rahulrc1127:~$ javac filemonitorwithlogger.java
filemonitorwithlogger.java:6: error: class FileMonitorWithLogger is public, should be declared in a file named FileMonitorWithLogger.java
public class FileMonitorWithLogger {
^
1 error
ubuntu@rahulrc1127:~$ nano filemonitorwithlogger.java
ubuntu@rahulrc1127:~$ javac filemonitorwithlogger.java
ubuntu@rahulrc1127:~$ java filemonitorwithlogger
Monitoring started on: /home/ubuntu/Documents/critical
Logging to: file_monitor_log.csv
ENTRY_CREATE: test.txt
ENTRY_MODIFY: test.txt
ENTRY_CREATE: .ransomwareencryptor.java.swp
ENTRY_MODIFY: .ransomwareencryptor.java.swp
ENTRY_CREATE: .ransomwareencryptor.java.swp
ENTRY_MODIFY: .ransomwareencryptor.java.swp
ENTRY_CREATE: ransomwareencryptor.java
ENTRY_MODIFY: ransomwareencryptor.java
ENTRY_CREATE: ransomwareencryptor.class
ENTRY_MODIFY: ransomwareencryptor.class
ENTRY_CREATE: .ransomwareencryptor.java.swp
ENTRY_MODIFY: .ransomwareencryptor.java.swp
ENTRY_CREATE: .ransomwareencryptor.java.swp
ENTRY_MODIFY: .ransomwareencryptor.java.swp
ENTRY_CREATE: ransomwaredecryptor.java
ENTRY_MODIFY: ransomwaredecryptor.java
ENTRY_CREATE: ransomwaredecryptor.class
ENTRY_MODIFY: ransomwaredecryptor.class
ENTRY_CREATE: .ransomwaredecryptor.java.swp
ENTRY_MODIFY: .ransomwaredecryptor.java.swp
ENTRY_CREATE: .ransomwaredecryptor.java.swp
ENTRY_MODIFY: .ransomwaredecryptor.java.swp
ENTRY_CREATE: ransomwaredecryptor.java
ENTRY_MODIFY: ransomwaredecryptor.java
ENTRY_CREATE: ransomwaredecryptor.class
ENTRY_MODIFY: ransomwaredecryptor.class
ENTRY_CREATE: .ransomwaredecryptor.java.swp
ENTRY_MODIFY: .ransomwaredecryptor.java.swp
ENTRY_CREATE: .ransomwaredecryptor.java.swp
ENTRY_MODIFY: .ransomwaredecryptor.java.swp
ENTRY_CREATE: ransomwaredecryptor.java
ENTRY_MODIFY: ransomwaredecryptor.java
ENTRY_CREATE: ransomwaredecryptor.class
ENTRY_MODIFY: ransomwaredecryptor.class
ENTRY_CREATE: .ransomwaredecryptor.java.swp
ENTRY_MODIFY: .ransomwaredecryptor.java.swp

```

The third step required designers to create FileMonitorLogger.java which served as a Java-based file monitoring system for detecting ransomware activities operating on the victim's system.

The monitor implements Java's WatchService API to track live changes which occur in the /home/ubuntu/Documents/critical sensitive directory. The program records three elements of file system changes — creation, modification, and removal — to file_monitor_log.csv as a structured log file.

The active monitoring component of the system checks for the creation of .enc files because this action indicates ransomware has encrypted files.

The system detects these events through its API it will create console warning alerts. The component actively behaves like a HIDS system while it also supports automation capabilities to launch a decryptor post-payment and to isolate the system.

Real-time ransomware detection operates similarly to OSSEC among other tools according to their detection methods.