SF Lab 5

Ponnanna A H (21CS02003)

25/03/2025

About the Assignment

This assignment focuses on simulating ransomware from a malware repository and analyzing its impact on system resources, including processes, directories, CPU usage, and memory consumption. Four ransomware variants—RedBoot, Dharma, GoldenEye, and WannaCry—were tested. Additionally, baseline system resource statistics were recorded prior to the attacks for comparison.

Monitoring System Resources

The following C program collects data on system resources such as CPU usage, memory availability, process count, and thread count. The data is stored in a CSV file for further analysis.

```c
#include <psapi.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <windows.h>

#define LOG_FILE "C:\\Users\\Student\\Desktop\\system_resources.csv"
#define SAMPLE_INTERVAL 2

void getCurrentTimestamp(char *buffer, size_t size) {
    time_t now = time(NULL);
    struct tm *localTime = localtime(&now);
    strftime(buffer, size, "%Y-%m-%d %H:%M:%S", localTime);
}

void initializeLogFile() {
    FILE *file = fopen(LOG_FILE, "w");
    if (file == NULL) {
        printf("Error: Unable to create log file at %s\n", LOG_FILE);
        exit(EXIT_FAILURE);
```

```
    }
    fprintf(file,
"Timestamp,CPU_Usage_Percent,Available_Memory_MB,Total_Memory_MB,Memory_Usage_P
ercent,Process_Count\n");
    fclose(file);
    printf("Log file initialized at: %s\n", LOG_FILE);
}

double getCpuUsage() {
    static ULARGE_INTEGER lastCPU, lastSysCPU, lastUserCPU;
    static int initialized = 0;
    FILETIME ftime, fsys, fuser;

    GetSystemTimeAsFileTime(&ftime);
    memcpy(&lastCPU, &ftime, sizeof(FILETIME));
    GetProcessTimes(GetCurrentProcess(), &ftime, &ftime, &fsys, &fuser);

    if (!initialized) {
        initialized = 1;
        memcpy(&lastSysCPU, &fsys, sizeof(FILETIME));
        memcpy(&lastUserCPU, &fuser, sizeof(FILETIME));
        return -1.0;
    }

    ULARGE_INTEGER now, sysCPU, userCPU;
    memcpy(&now, &ftime, sizeof(FILETIME));
    memcpy(&sysCPU, &fsys, sizeof(FILETIME));
    memcpy(&userCPU, &fuser, sizeof(FILETIME));

    double percent = (double)((sysCPU.QuadPart - lastSysCPU.QuadPart) +
(userCPU.QuadPart - lastUserCPU.QuadPart)) /
                     (now.QuadPart - lastCPU.QuadPart) * 100;

    lastSysCPU = sysCPU;
    lastUserCPU = userCPU;
    lastCPU = now;

    return percent;
}

void getMemoryUsage(double *availableMemoryMB, double *totalMemoryMB) {
    MEMORYSTATUSEX memInfo;
```

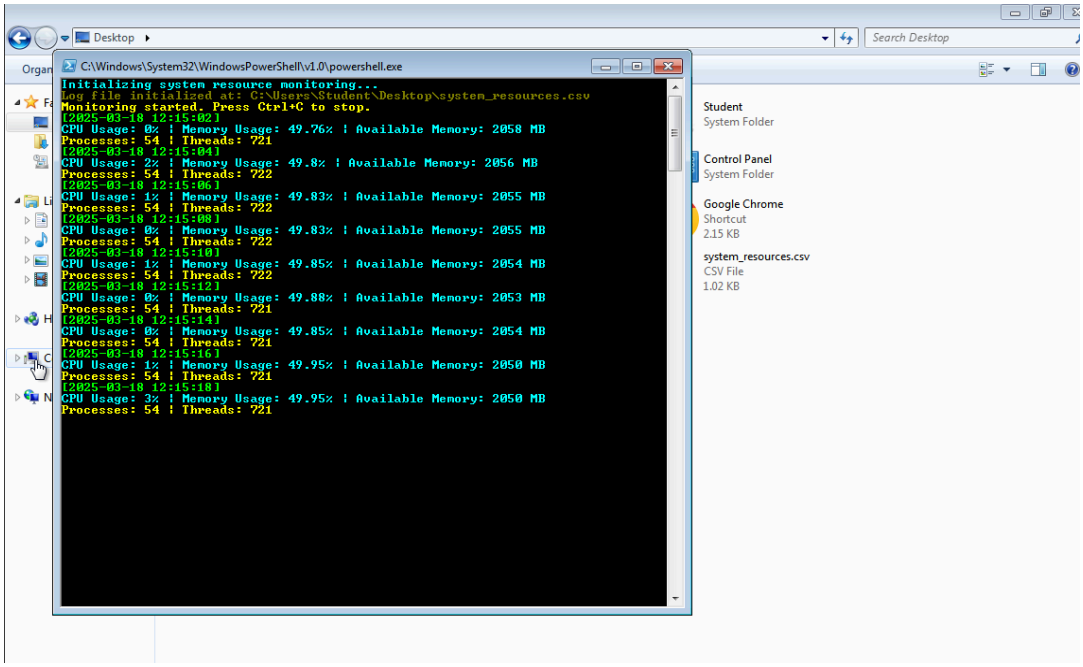
```
    memInfo.dwLength = sizeof(MEMORYSTATUSEX);
    GlobalMemoryStatusEx(&memInfo);

    *totalMemoryMB = memInfo.ullTotalPhys / (1024.0 * 1024.0);
    *availableMemoryMB = memInfo.ullAvailPhys / (1024.0 * 1024.0);
}

void getProcessCount(int *processCount) {
    DWORD processes[1024], cbNeeded;
    if (!EnumProcesses(processes, sizeof(processes), &cbNeeded)) {
        printf("Error: Unable to enumerate processes.\n");
        exit(EXIT_FAILURE);
    }

    *processCount = cbNeeded / sizeof(DWORD);
}

int main() {
    initializeLogFile();

    while (1) {
        FILE *file = fopen(LOG_FILE, "a");
        if (file == NULL) {
            printf("Error: Unable to open log file at %s\n", LOG_FILE);
            exit(EXIT_FAILURE);
        }

        char timestamp[20];
        getCurrentTimestamp(timestamp, sizeof(timestamp));

        double cpuUsage = getCpuUsage();
        double availableMemoryMB = 0.0;
        double totalMemoryMB = 0.0;
        getMemoryUsage(&availableMemoryMB, &totalMemoryMB);

        int processCount = 0;
        getProcessCount(&processCount);

        fprintf(file,
                "%s,%f,%f,%f,%f,%d\n",
                timestamp,
                cpuUsage,
```

```
                availableMemoryMB,
                totalMemoryMB,
                ((totalMemoryMB - availableMemoryMB) / totalMemoryMB) * 100,
                processCount);

        fclose(file);
        Sleep(SAMPLE_INTERVAL * 1000); // Interval in milliseconds
    }

    return 0;
}
```

Ransomware Analysis

Pre-Attack Observations

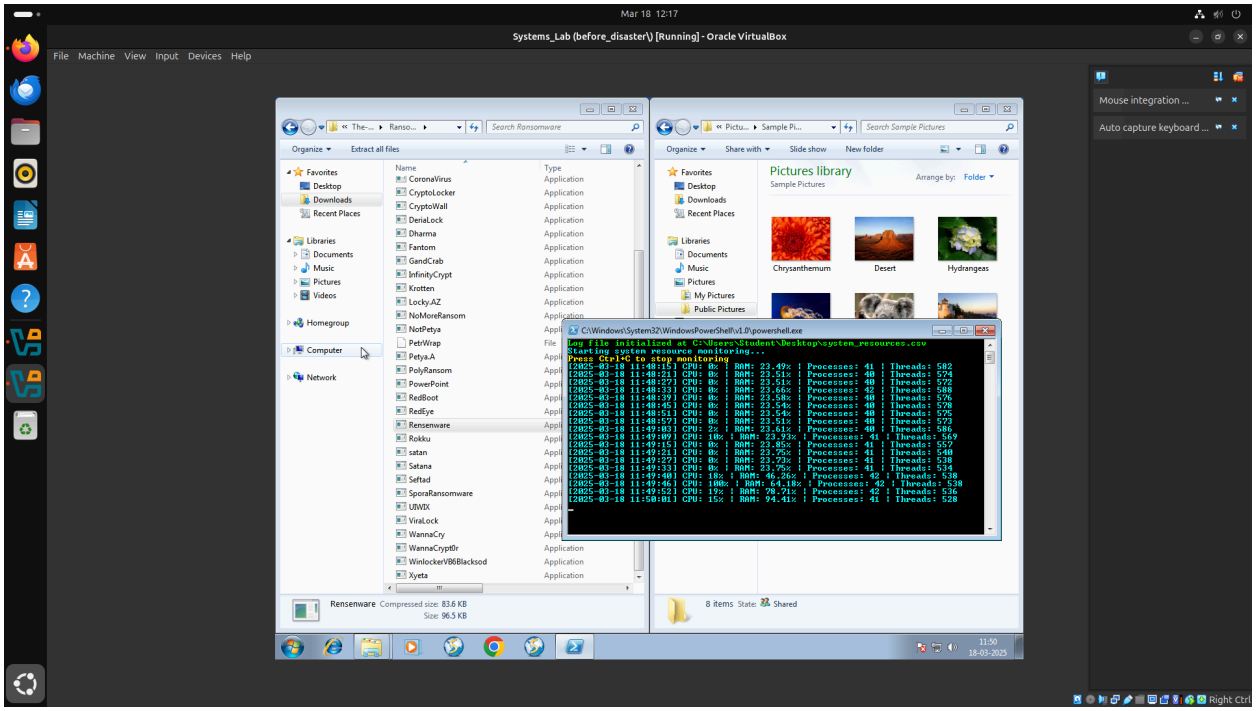
- System resources remain stable before any ransomware is executed.
- Normal levels of CPU and RAM usage are observed.
- No significant changes in processes or directories.



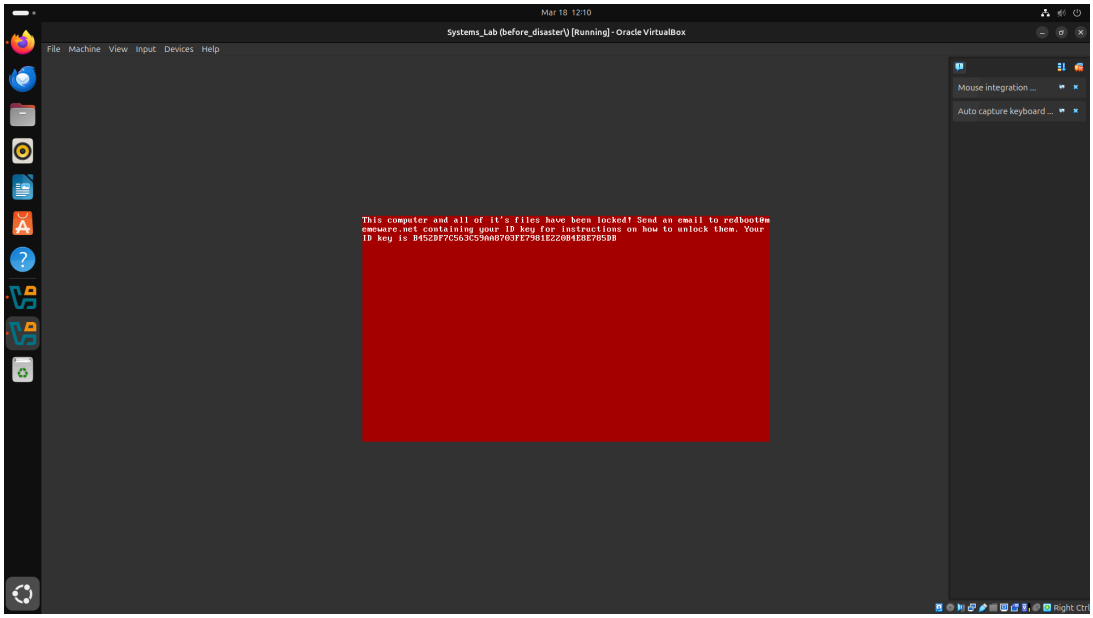
Post-Attack Observations

The following ransomware variants were tested on Windows 7 in Oracle VirtualBox:

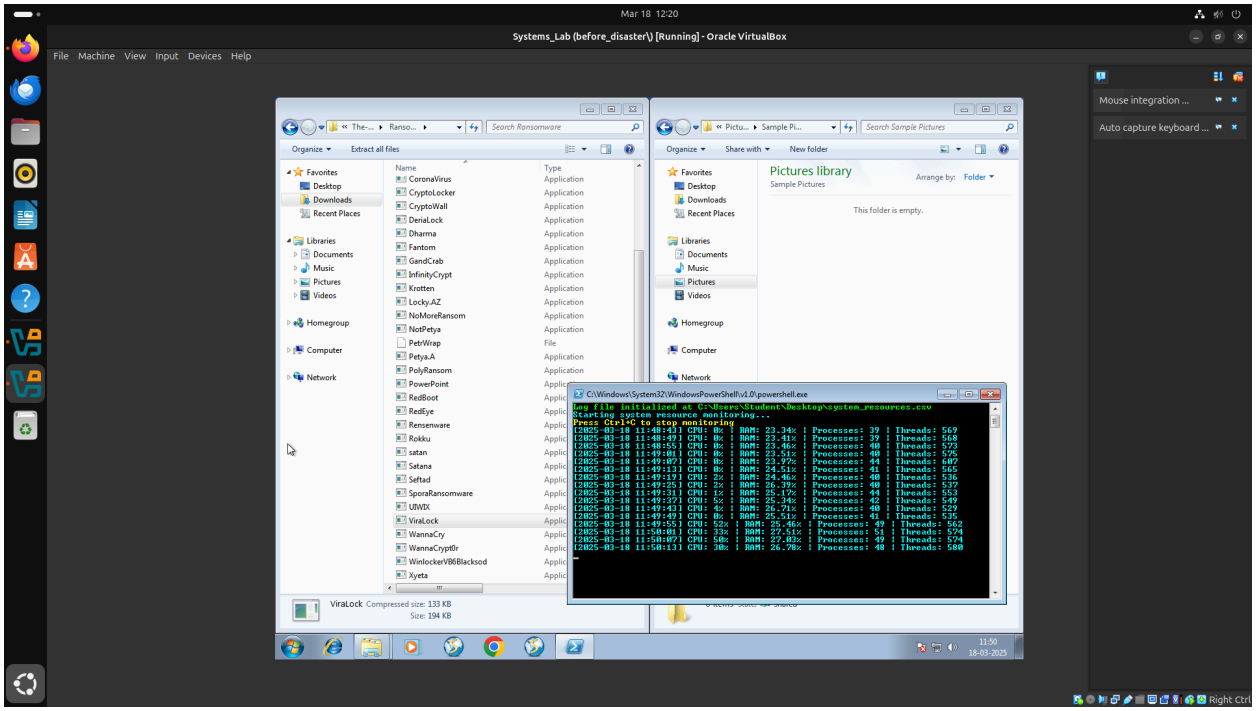
- **Rensenware**
    - **Behavior:** Causes resource bottlenecks.
    - **Impact:** High CPU and RAM usage during execution; minimal changes to files.



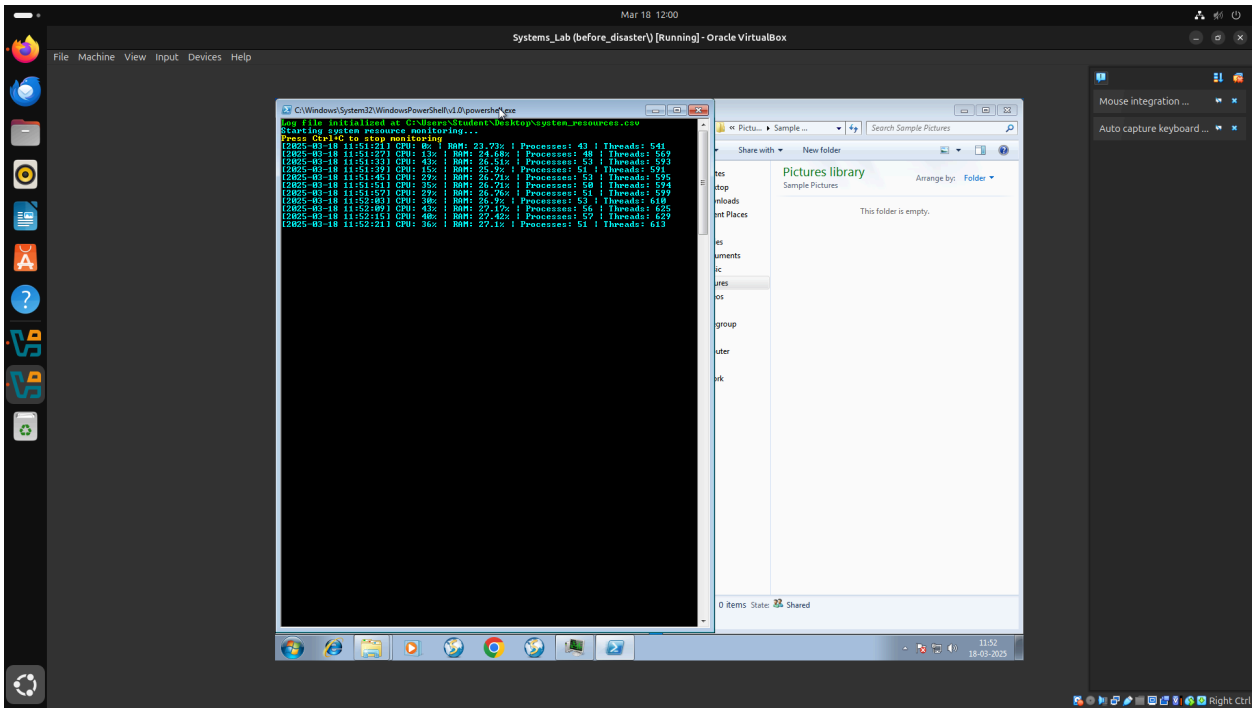
- **RedBoot**
    - **Behavior:** Alters the Master Boot Record (MBR), making the system unbootable.
    - **Impact:** Increased disk activity and system instability.



- **Viralock**
  - **Behavior:** Encrypts all system files and hides them from the user.
  - **Impact:** High CPU usage and moderate RAM consumption during encryption; significant file changes.



- **Polyransom**
    - **Behavior:** Encrypts files and alters folder structures.
    - **Impact:** Moderate CPU and RAM usage; noticeable changes in directory structure.



Summary of all Ransomwares

State/Ransomware	CPU Usage	RAM Usage	Process Count	Folder Changes
Before Disaster	Stable	Stable	Normal	None
Rensenware	High	High	Stable	Minimal
Redboot	Moderate	Low	Stable	Minimal
Viralock	High	Moderate	Increased	Significant
Polyransom	Moderate	Moderate	Increased	SIgnificant

Conclusion

The analysis highlights the impact of ransomware attacks on system resources such as CPU usage and memory consumption. Baseline statistics provide a clear point of comparison for identifying abnormal behavior caused by ransomware.